libEnsemble: A Library for Managing Ensembles of Calculations

Stephen Hudson Jeffrey Larson Stefan Wild

Argonne National Laboratory

July 13, 2018

What is `libEnsemble`

- `libEnsemble` is a software library to coordinate the concurrent evaluation of ensembles of calculations.

What is `libEnsemble`

- `libEnsemble` is a software library to coordinate the concurrent evaluation of ensembles of calculations.

- `libEnsemble` uses a manager to allocate work to various workers.

What is `libEnsemble`

- `libEnsemble` is a software library to coordinate the concurrent evaluation of ensembles of calculations.
- `libEnsemble` uses a manager to allocate work to various workers.
- A `libEnsemble` worker is the smallest indivisible unit to perform

libEnsemble requires of the user

`gen_f`: Generates inputs to `sim_f` and `alloc_f`.

libEnsemble requires of the user

gen_f: Generates inputs to sim_f and alloc_f.

sim_f: Evaluates a simulation (i.e., user-defined function) using input defined by gen_f.

libEnsemble requires of the user

`gen_f`: Generates inputs to `sim_f` and `alloc_f`.

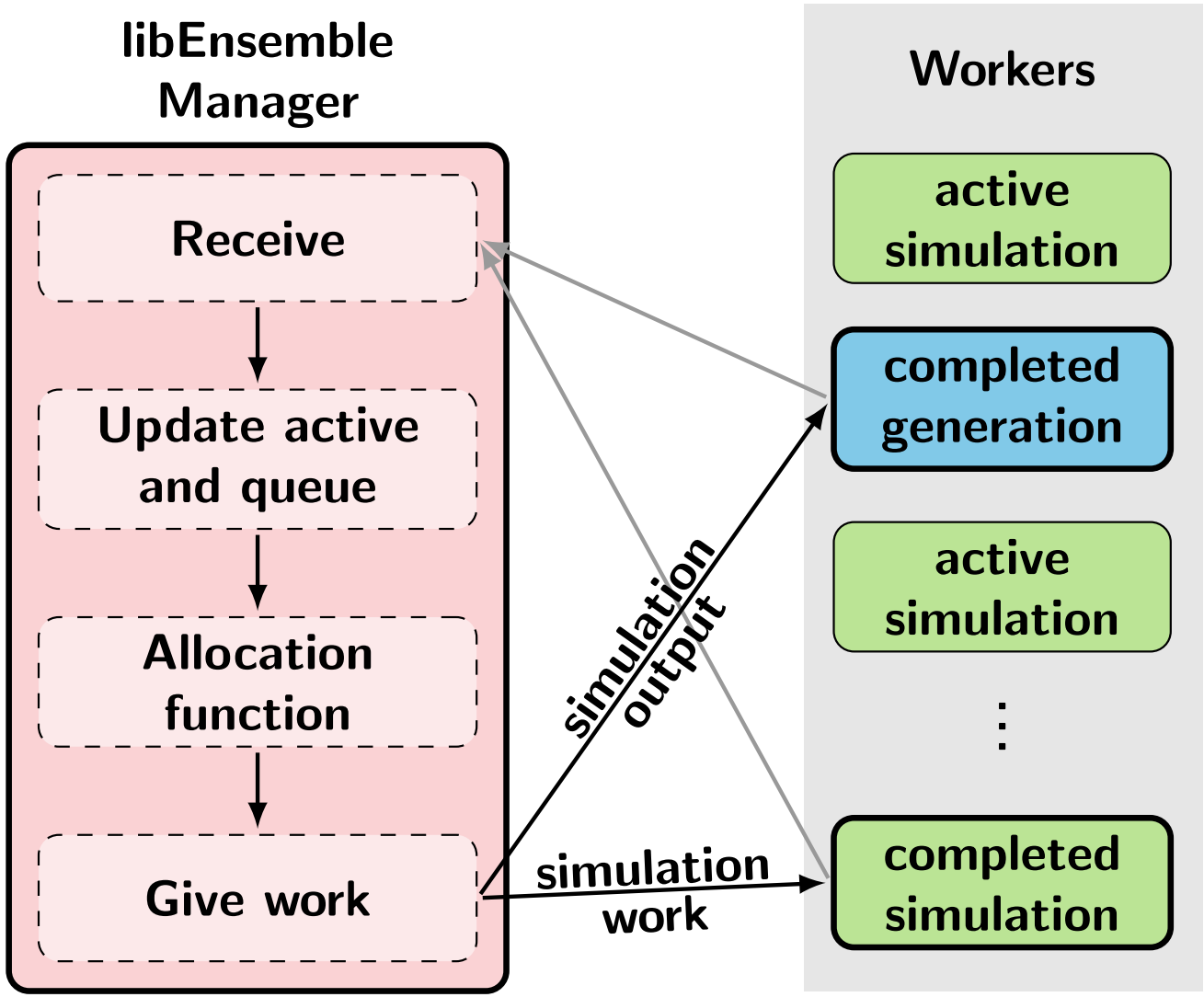
`sim_f`: Evaluates a simulation (i.e., user-defined function) using input defined by `gen_f`.

`alloc_f`: Decides whether (or not) `sim_f` or `gen_f` should be called (and with what input/resources) as workers become available.

`libEnsemble` dependencies

- Python 2.7, 3.4 or above.
- An MPI 1.x/2.x/3.x implementation (e.g., MPICH or OpenMPI) built with shared/dynamic libraries.
- mpi4py v2.0.0 or above.
- NumPy v1.10 or above.

libEnsemble overview



Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation
- Ability to recover hardware resources from `libEnsemble`-/generation-/simulation-terminated calculations

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation
- Ability to recover hardware resources from `libEnsemble`-/generation-/simulation-terminated calculations
- (*) Simulation/generation checkpoint and restart

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation
- Ability to recover hardware resources from `libEnsemble`-/generation-/simulation-terminated calculations
- (*) Simulation/generation checkpoint and restart
- (*) Execution on multiple LCFs

Possible user requirements of `libEnsemble`

- `sim_f`/`gen_f` calculations can employ/access parallel resources
  - This places requirements on user's environment and simulation/generation function
- Maintenance of calculation history and performance measures
- Termination of unresponsive simulation/generation calculations
- Termination based on intermediate simulation/generation output
- Nonfatal handling (i.e., graceful degradation/fail soft) of failed simulation/generation calculation
- Ability to recover hardware resources from `libEnsemble`-/generation-/simulation-terminated calculations
- (*) Simulation/generation checkpoint and restart
- (*) Execution on multiple LCFs
- (*) Thousands of concurrent workers

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel.

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel.
- Add another level of parallelism to a simulation that no longer scales

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel.
- Add another level of parallelism to a simulation that no longer scales
- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work.

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel.
- Add another level of parallelism to a simulation that no longer scales
- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work.
- Don't have to write your own kills, just complete `libEnsemble` templates.

Why would you want to use `libEnsemble`?

- Easily take serial code and start running in parallel.
- Add another level of parallelism to a simulation that no longer scales
- Don't have to write your own tracking code, `libEnsemble` will tell you which runs fail and start other work.
- Don't have to write your own kills, just complete `libEnsemble` templates.
- Want to add concurrency to a generator (e.g., multiple local optimizers.)

Why `libEnsemble` and not...?

Dakota:

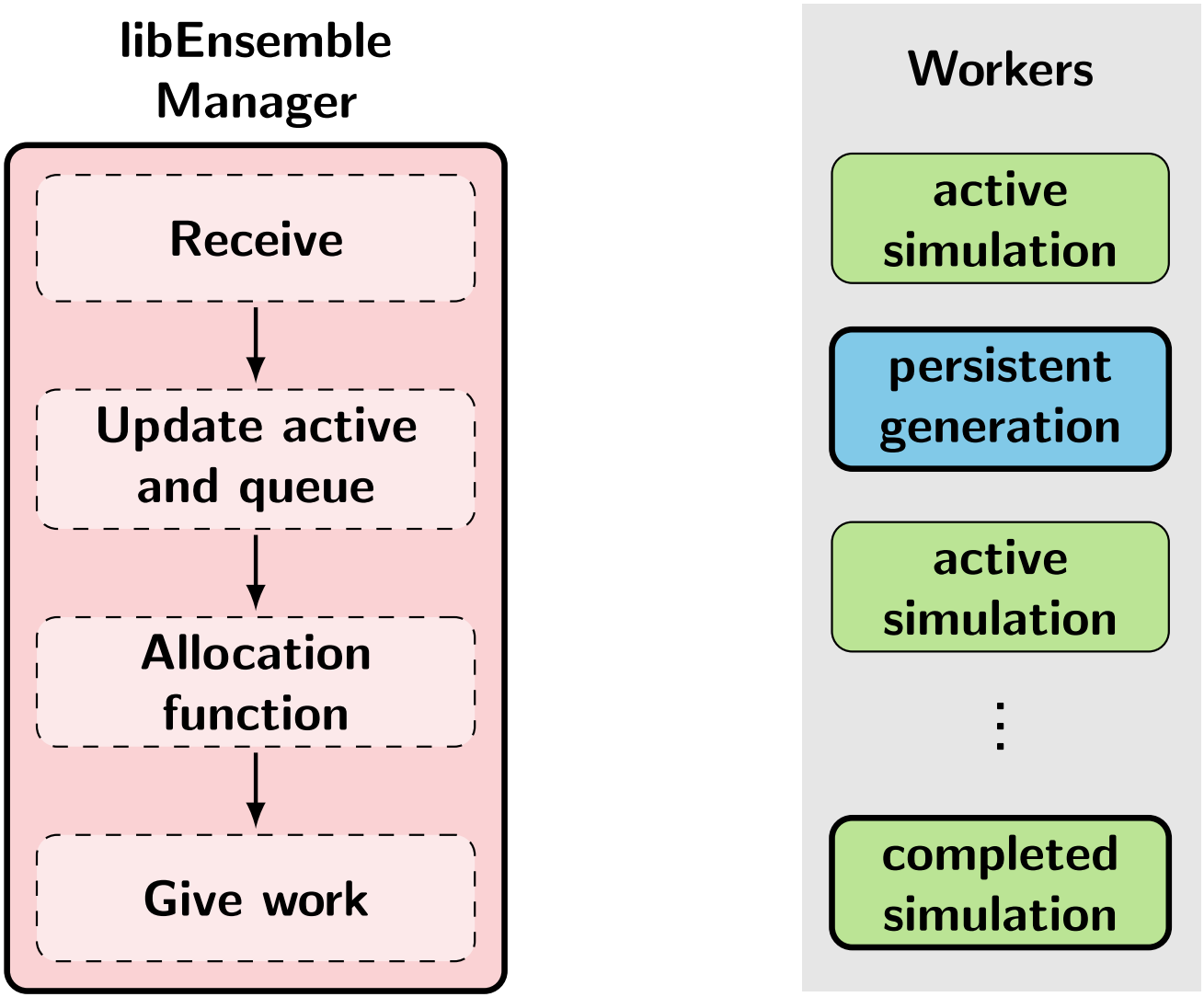
- Few simulation optimization generators.
- C++

Why `libEnsemble` and not...?

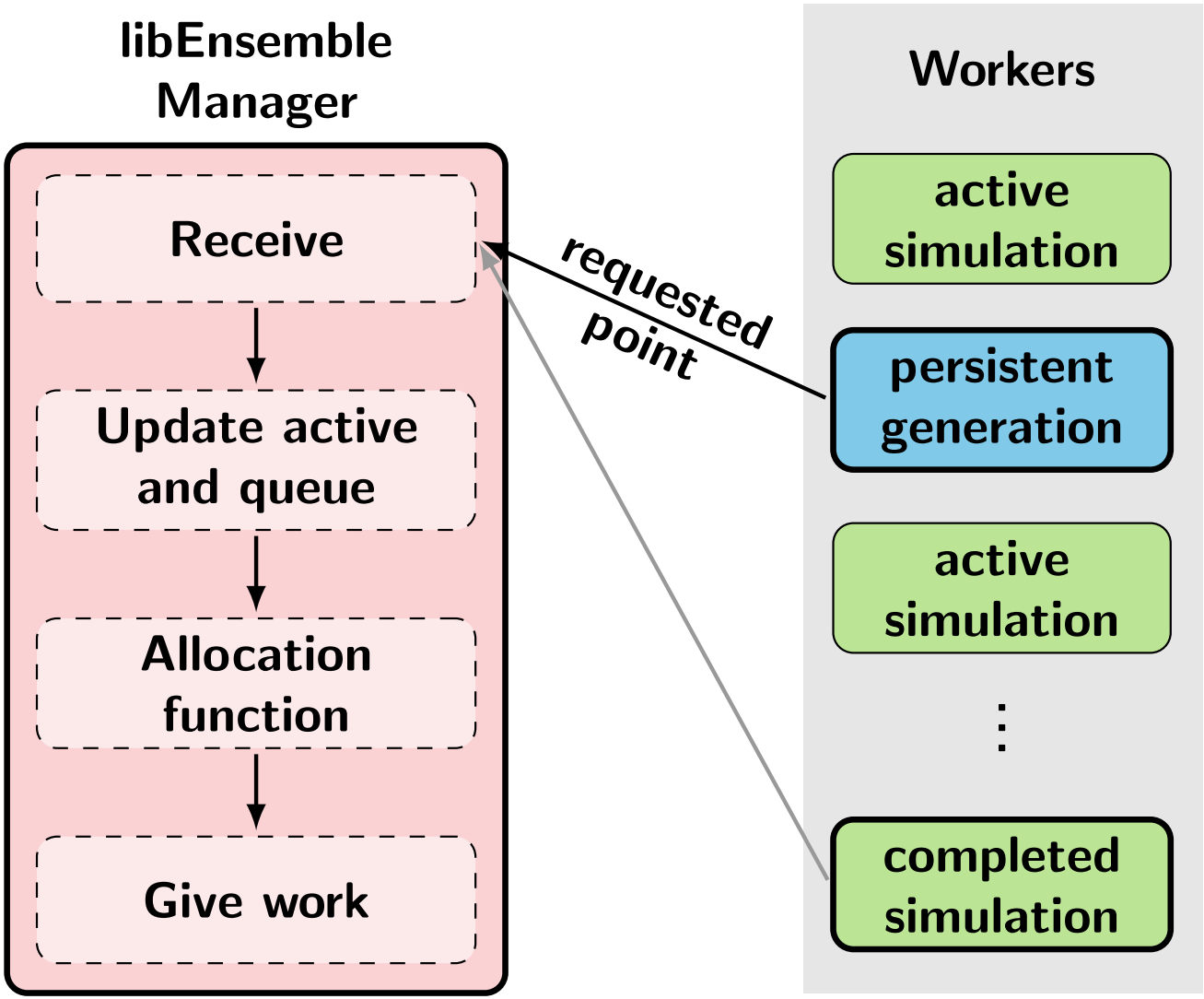
Dakota:
- Few simulation optimization generators.
- C++

Swift: (the parallel scripting language)
- "Can run million programs, thousands at a time, launching hundreds per second."
- Require writing your generators in Swift's scripting language.
- Difficult to tightly couple generation of inputs and future/active running simulations.

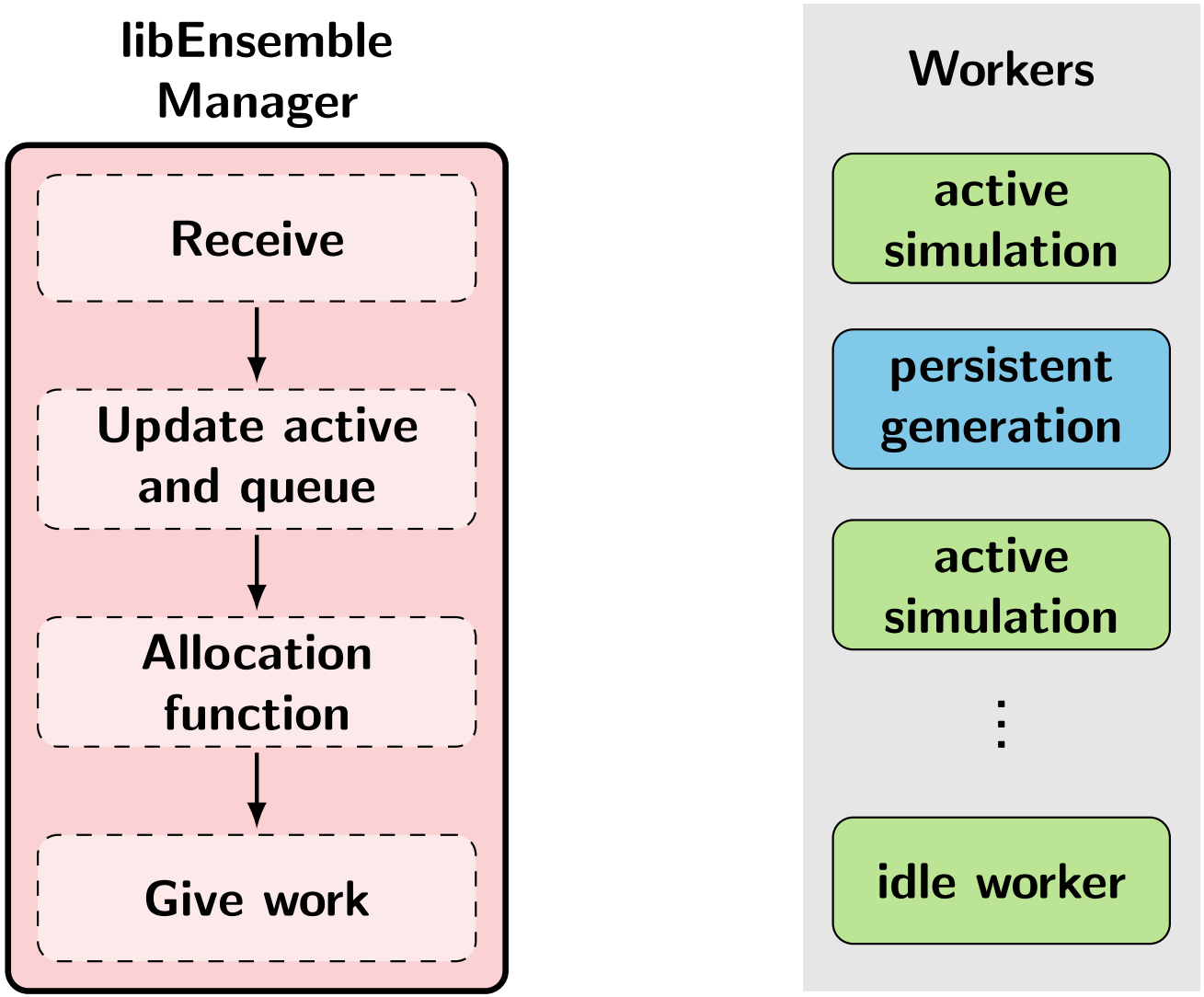
libEnsemble overview



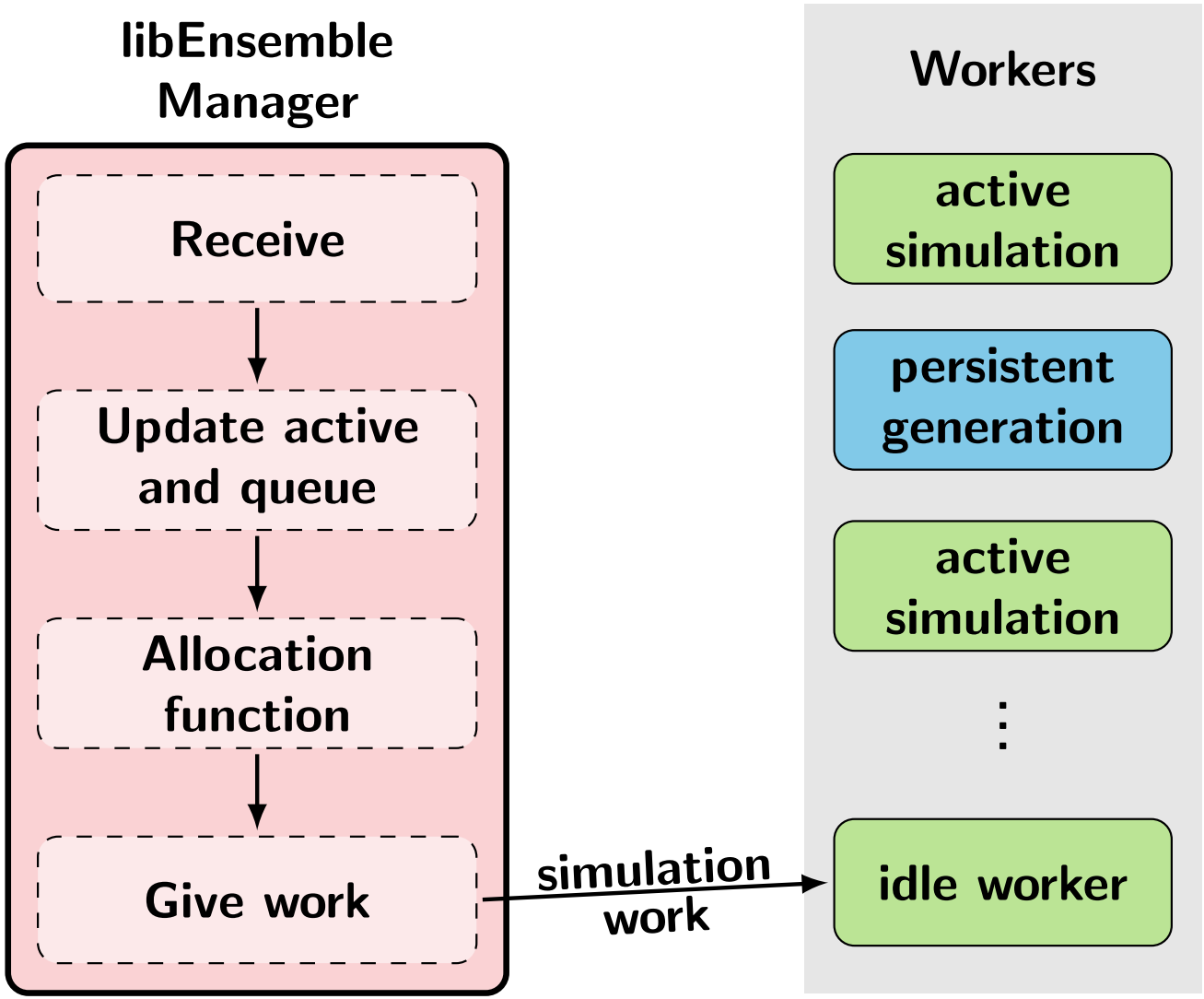
libEnsemble overview



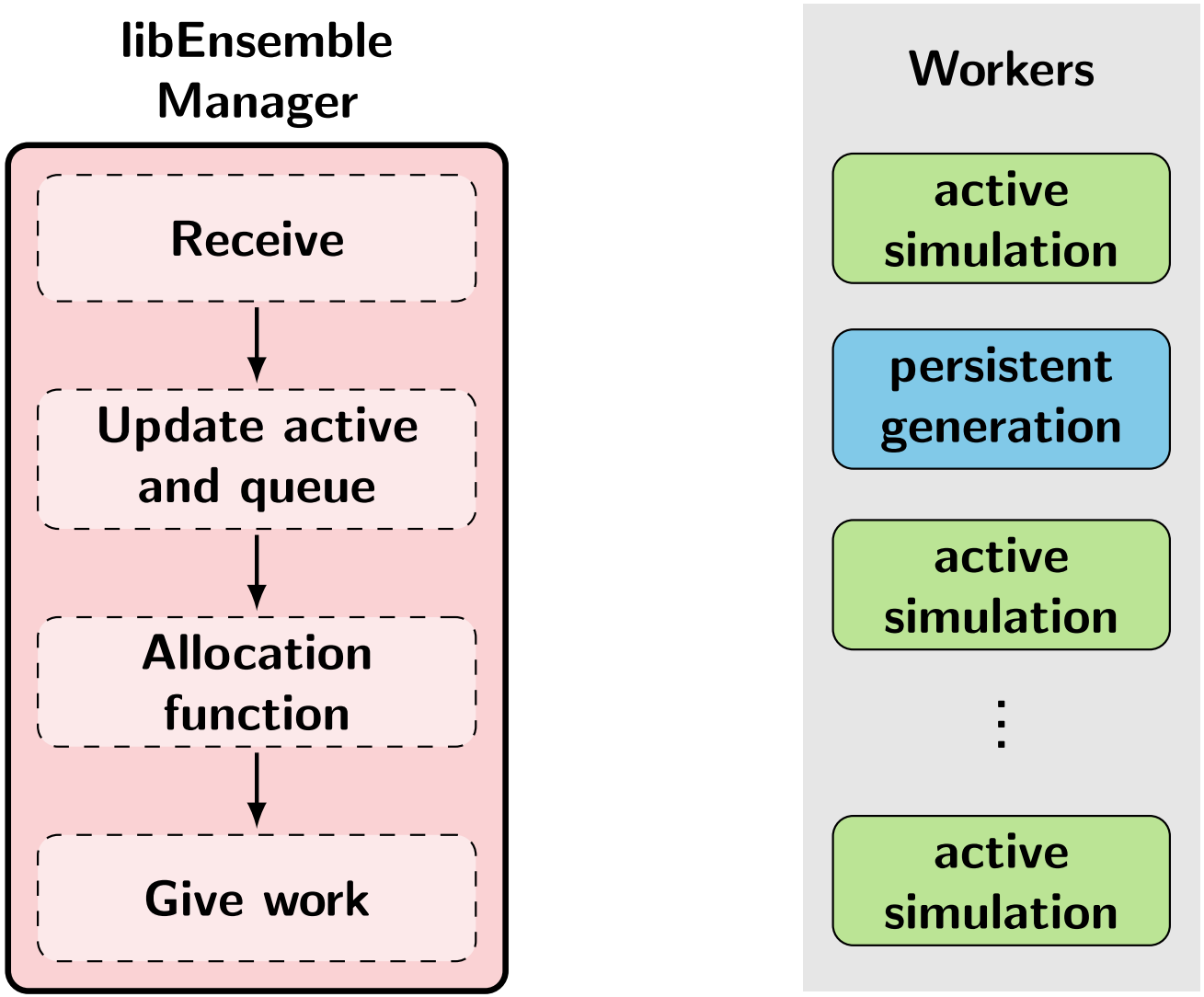
libEnsemble overview



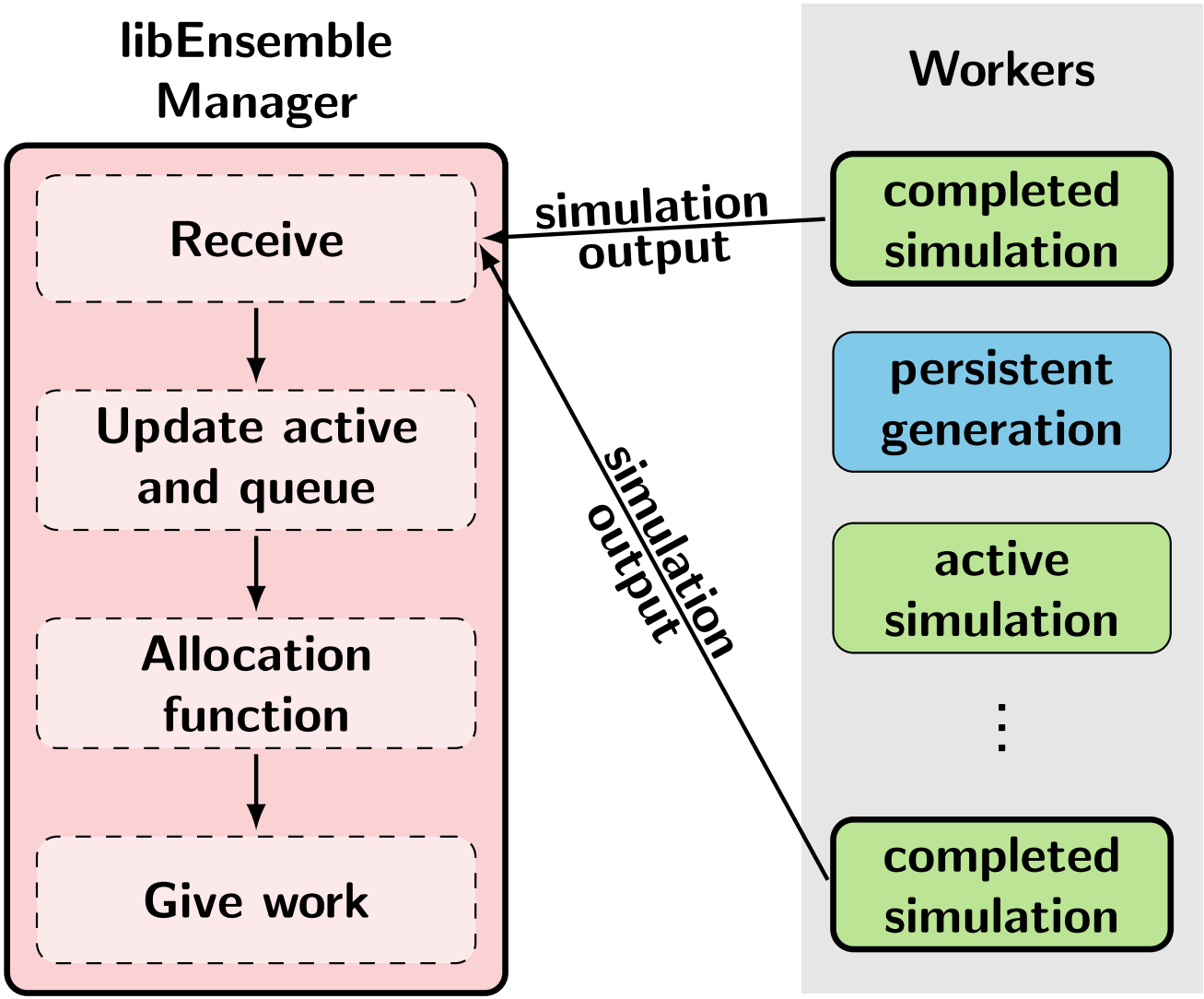
libEnsemble overview



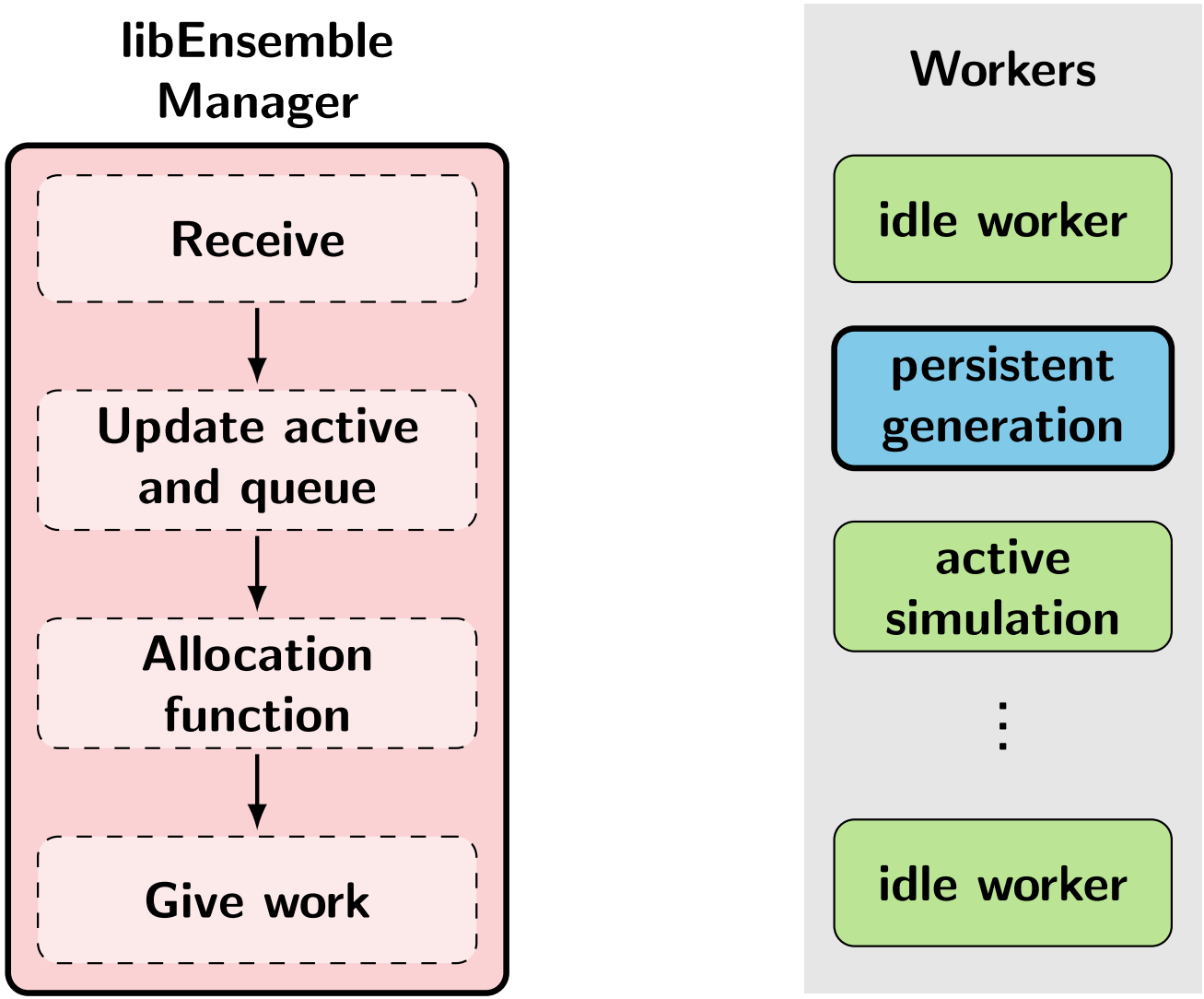
libEnsemble overview



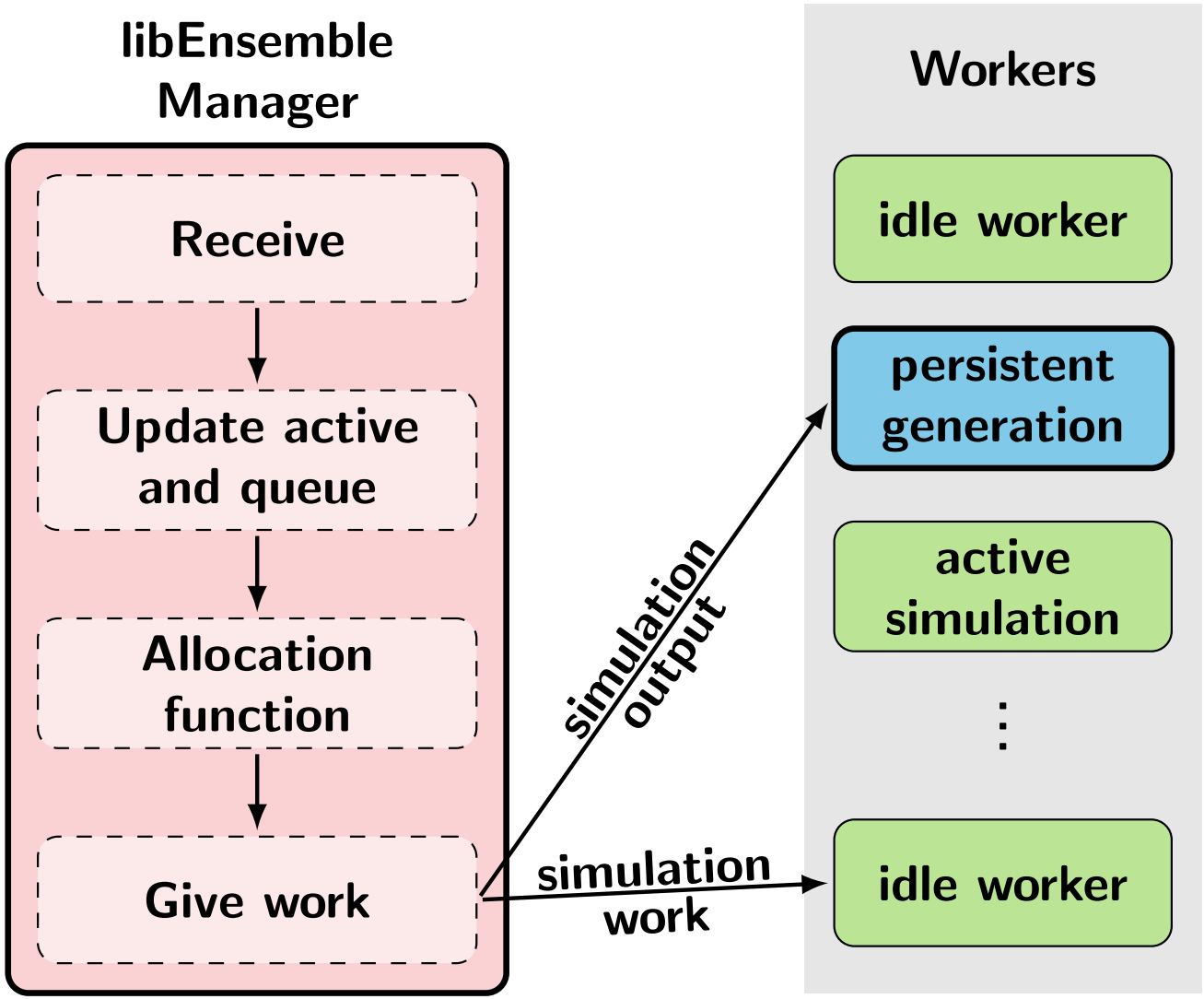
libEnsemble overview



libEnsemble overview



libEnsemble overview



Timing `libEnsemble` overhead

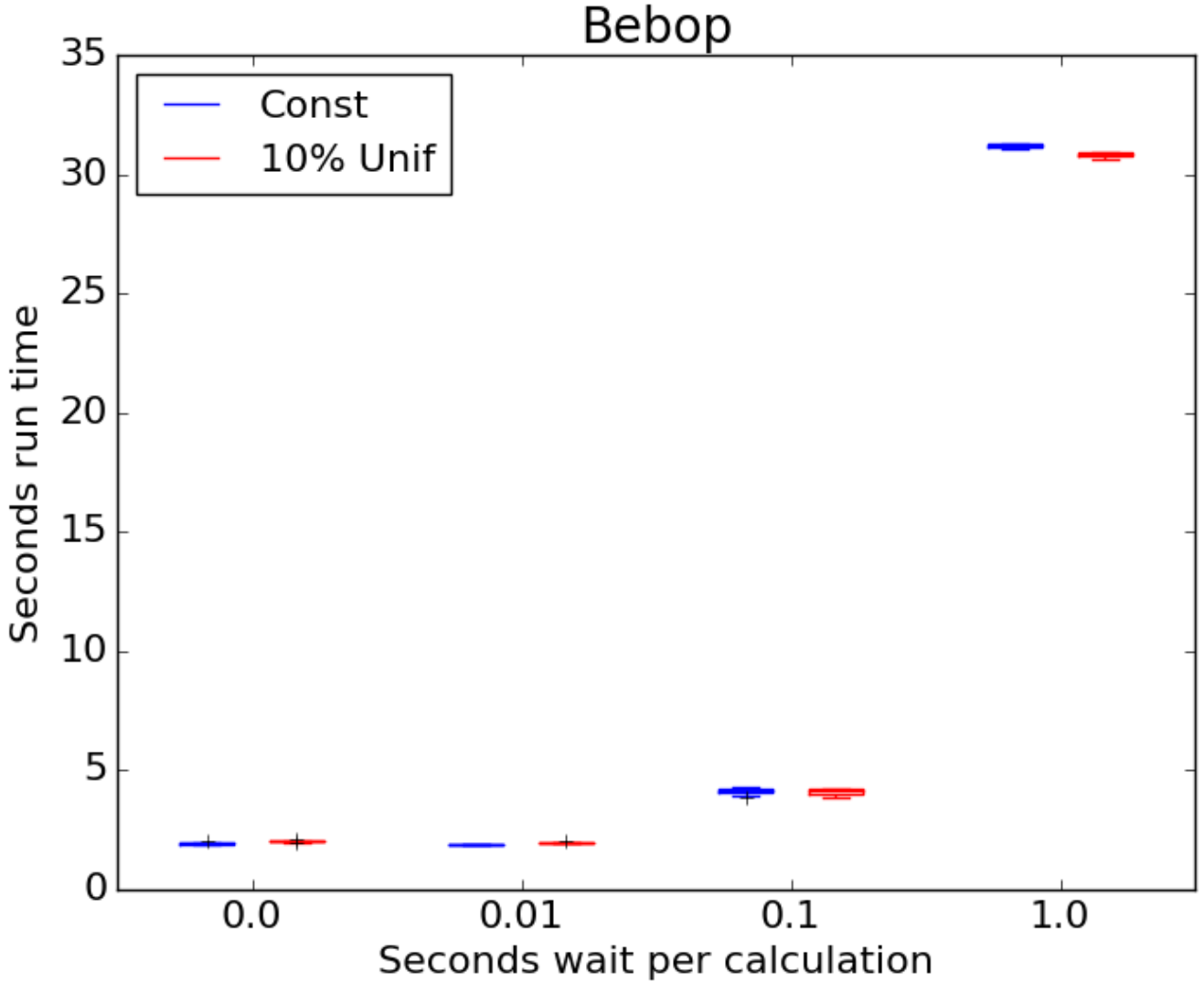
Time for `libEnsemble` to sample/evaluation 30$\times$(workers) points

Tons of system-dependent caveats

Timing `libEnsemble` overhead

Time for `libEnsemble` to sample/evaluation 30×(workers) points

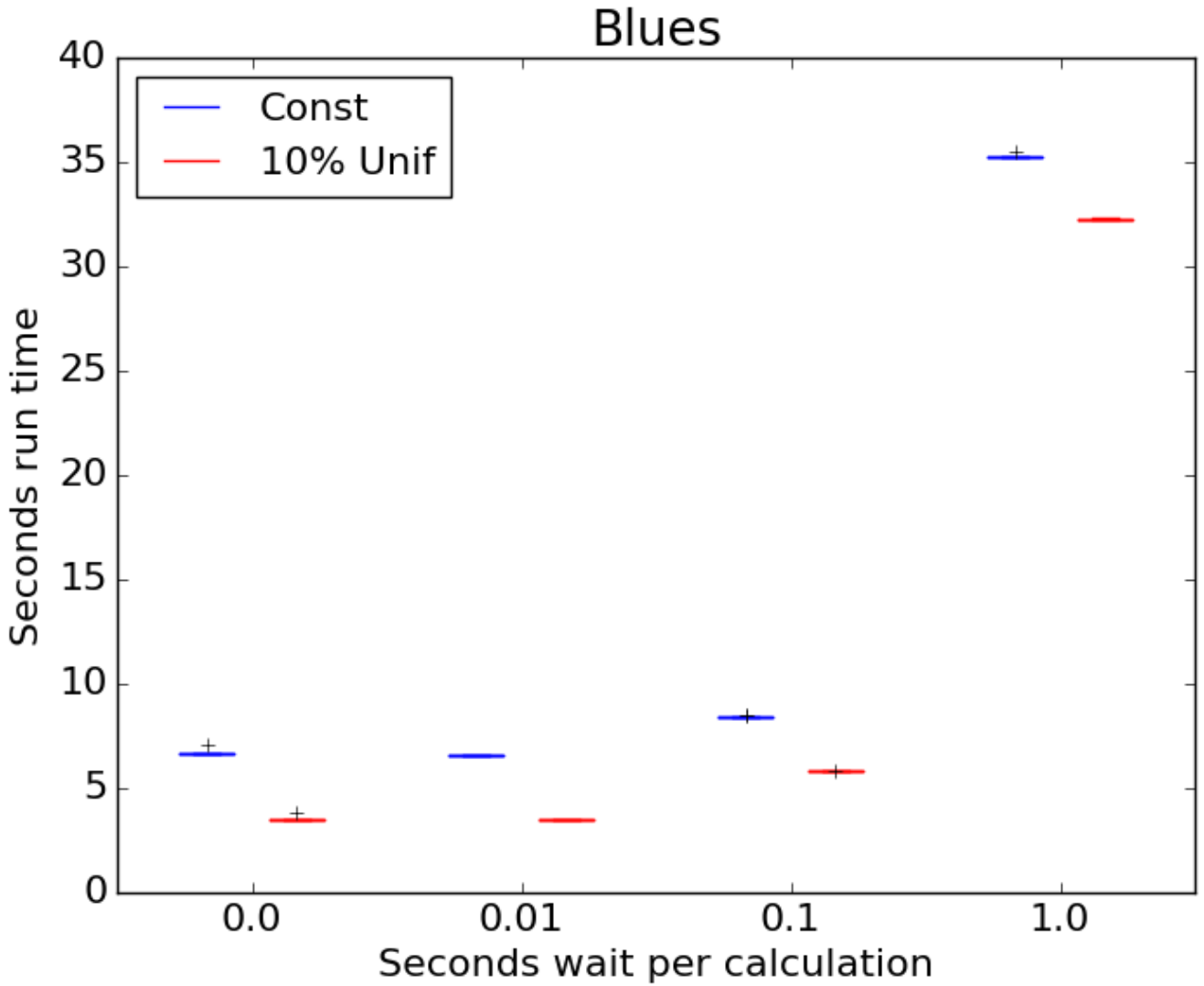
32 nodes × 36 cores = 1152-1 workers



Timing `libEnsemble` overhead

Time for `libEnsemble` to sample/evaluation 30×(workers) points

64 nodes × 16 cores = 1024-1 workers



Use cases

- A user wants to optimize a function that depends on a simulation.
- The simulation is already using parallel resources, but not a large fraction of some computer.
- `libEnsemble` can coordinate the concurrent evaluation of the simulation `sim_f` at various parameter values and `gen_f` would return candidate parameter values (possibly after each `sim_f` output).

Use cases

- A user has a `gen_f` that produces different meshes to be used within a `sim_f`.
- Given the `sim_f` output, `gen_f` will refine a mesh or produce a new mesh.
- `libEnsemble` can ensure that the calculated meshes can be used by multiple simulations without requiring movement of data.

Use cases

- A user wants to evaluate a simulation `sim_f` at parameters sampled from a set of parameter values.
- Many parameter sets will cause the simulation to fail.
- `libEnsemble` can stop unresponsive evaluations, and recover computational resources for future evaluations.
- `gen_f` can update the sampling after discovering regions where evaluations of `sim_f` fail.

Use cases

- A user has a simulation `sim_f` that requires calculating multiple expensive quantities, some of which depend on other quantities.
- `libEnsemble` can observe intermediate quantities in order to stop related calculations and preempt future calculations associated with a poor parameter values.

Use cases

- A user wishes to identify multiple local optima for a `sim_f`.
- `libEnsemble` can use the points from the APOSMM `gen_f` to identify optima

Use cases

- A user wishes to identify multiple local optima for a `sim_f`.
- `libEnsemble` can use the points from the APOSMM `gen_f` to identify optima

Naturally, combinations of use cases is supported as well.

Problem setup

- We want to identify distinct, "high-quality", local minimizers of

$$\begin{aligned} \text{minimize } & f(x) \\ & l \leq x \leq u \\ & x \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective.

Problem setup

- We want to identify distinct, "high-quality", local minimizers of

$$\begin{aligned} \text{minimize } & f(x) \\ & l \leq x \leq u \\ & x \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective.
- Derivatives of f may or may not be available.

Problem setup

- We want to identify distinct, "high-quality", local minimizers of

$$\begin{aligned} \text{minimize } & f(x) \\ l \leq & \, x \leq u \\ & x \in \mathbb{R}^n \end{aligned}$$

- High-quality can be measured by more than the objective.
- Derivatives of f may or may not be available.
- The simulation f is likely using parallel resources, but it does not utilize the entire machine.

Global optimization is difficult

Theorem (Törn and Žilinskas, *Global Optimization*, 1989)

An algorithm converges to the global minimum of any continuous f on a domain $\mathcal{D}$ if and only if the algorithm generates iterates that are dense in $\mathcal{D}$.

Global optimization is difficult

Theorem (Törn and Žilinskas, *Global Optimization*, 1989)

An algorithm converges to the global minimum of any continuous f on a domain $\mathcal{D}$ if and only if the algorithm generates iterates that are dense in $\mathcal{D}$.

The theory can be more than merely checking that a method generates iterates which are dense in the domain.

Multistart Methods

Given some local optimization routine $\mathcal{L}$:

Algorithm 1: General Multistart

for $k = 1, 2, \ldots$ **do**
- Evaluate f at N points drawn from $\mathcal{D}$
- Start $\mathcal{L}$ at some set (possibly empty) of previously evaluated points

Multistart Methods

Given some local optimization routine $\mathcal{L}$:

Algorithm 1: General Multistart

for $k = 1, 2, \ldots$ **do**
  Evaluate f at N points drawn from $\mathcal{D}$
  Start $\mathcal{L}$ at some set (possibly empty) of previously evaluated points

- Get to use problem specific local optimization routines.
- Possibly multiple levels of parallelism (objective, local method, global method); $\mathcal{L}$ may involve many sequential evaluations of f...

Multistart Methods

Given some local optimization routine $\mathcal{L}$:

Algorithm 1: General Multistart

for $k = 1, 2, \ldots$ **do**
  Evaluate f at N points drawn from $\mathcal{D}$
  Start $\mathcal{L}$ at some set (possibly empty) of previously evaluated points

- Get to use problem specific local optimization routines.
- Possibly multiple levels of parallelism (objective, local method, global method); $\mathcal{L}$ may involve many sequential evaluations of f...

- Which points should start runs?
- If resources are limited, how should points from each run receive priority?

Multistart Methods

Given some local optimization routine $\mathcal{L}$:

Algorithm 1: General Multistart

for $k = 1, 2, \ldots$ **do**
  Evaluate f at N points drawn from $\mathcal{D}$
  Start $\mathcal{L}$ at some set (possibly empty) of previously evaluated points

- Get to use problem specific local optimization routines.
- Possibly multiple levels of parallelism (objective, local method, global method); $\mathcal{L}$ may involve many sequential evaluations of f...
- Which points should start runs?
- If resources are limited, how should points from each run receive priority?
- Ideally, only one run is started for each minima.
- Exploring by sampling. Refining with $\mathcal{L}$.

Multi-Level Single Linkage

- $f \in C^2$, with local minima in the interior of $\mathcal{D}$, and the distance between these minima is bounded away from zero.
- $\mathcal{L}$ is strictly descent and converges to a minimum (not a stationary point).
- $$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\Gamma\left(1 + \frac{n}{2}\right) \mathrm{vol}\left(\mathcal{D}\right) \frac{\sigma \log kN}{kN}} \tag{1}$$

Theorem

If $r_k \to 0$, all local minima will be found almost surely.

Multi-Level Single Linkage

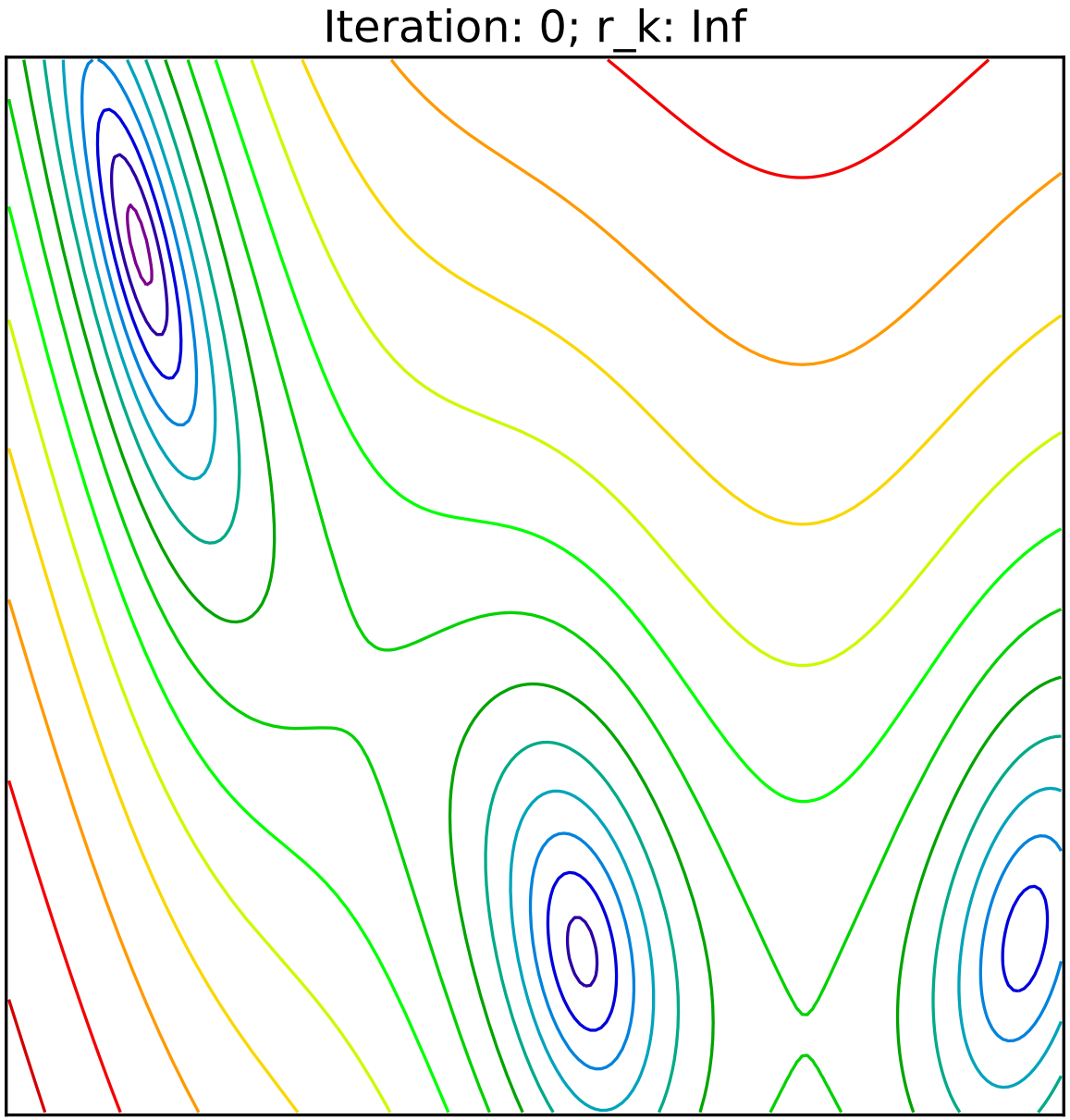
- $f \in C^2$, with local minima in the interior of $\mathcal{D}$, and the distance between these minima is bounded away from zero.
- $\mathcal{L}$ is strictly descent and converges to a minimum (not a stationary point).
- $$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\Gamma\left(1 + \frac{n}{2}\right) \mathrm{vol}\left(\mathcal{D}\right) \frac{\sigma \log kN}{kN}} \tag{1}$$

Theorem

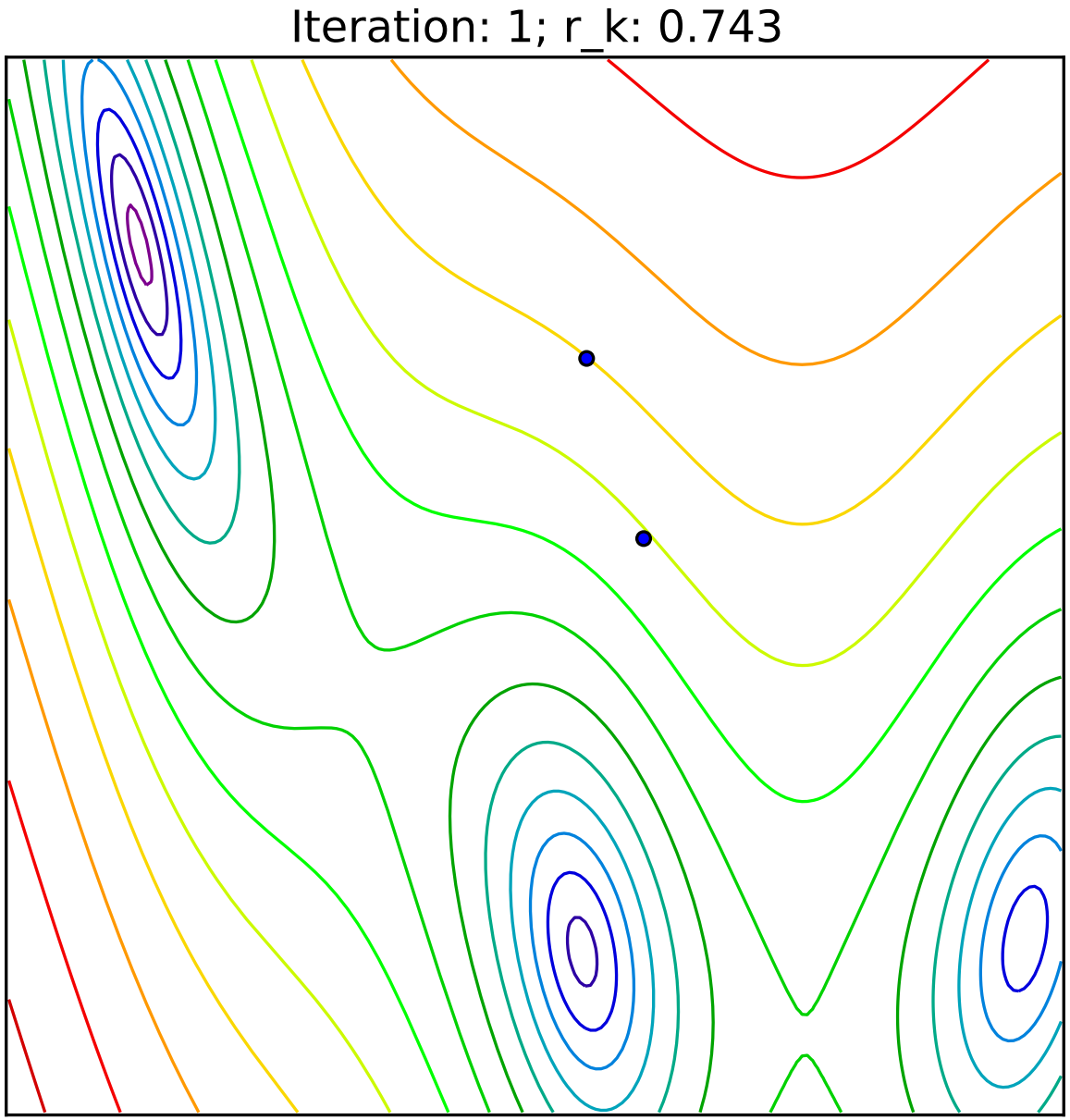
If $r_k \to 0$, all local minima will be found almost surely.

If r_k is defined by (1) with $\sigma > 4$, even if the sampling continues forever, the total number of local searches started is finite almost surely.

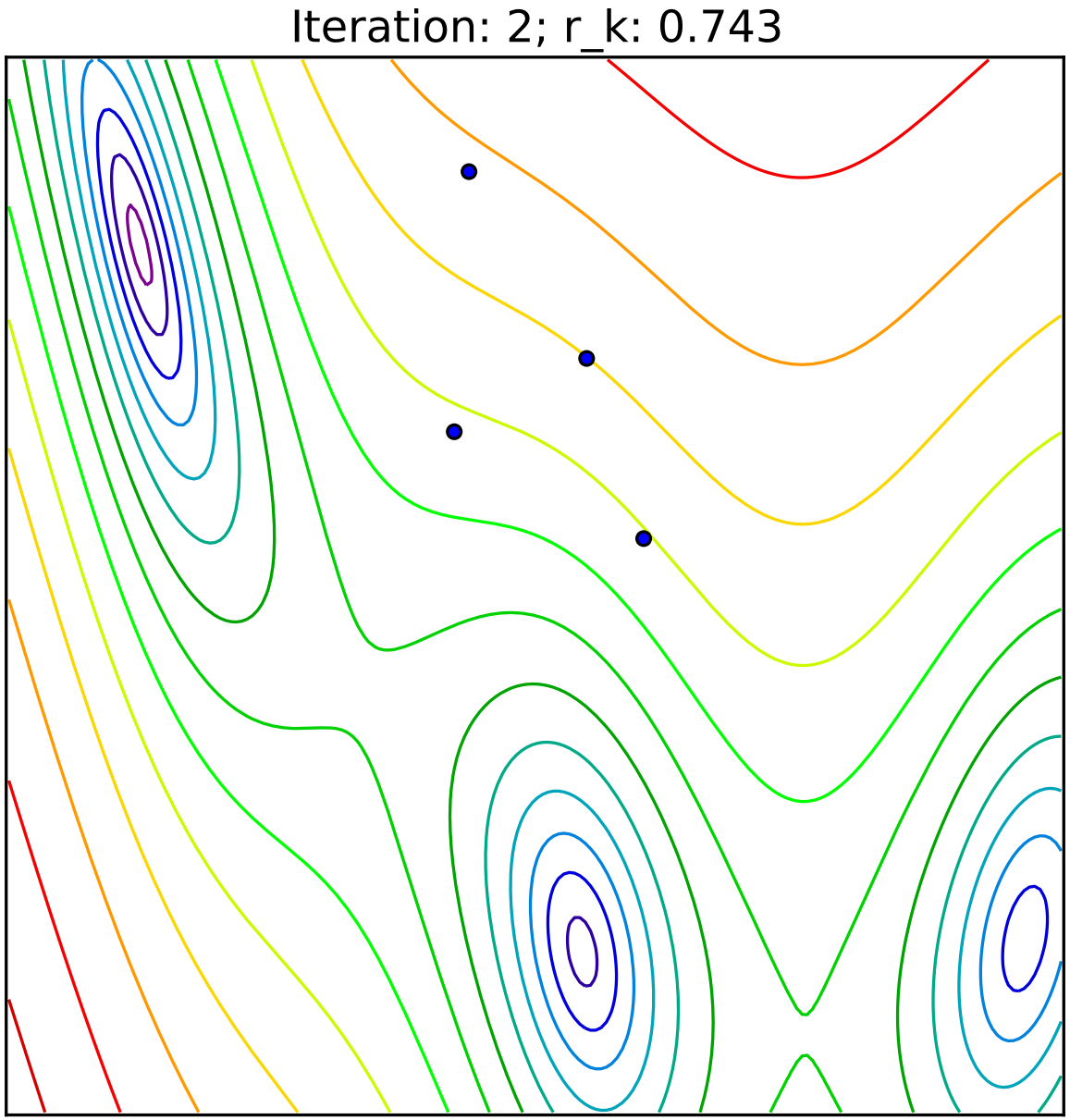
APOSMM



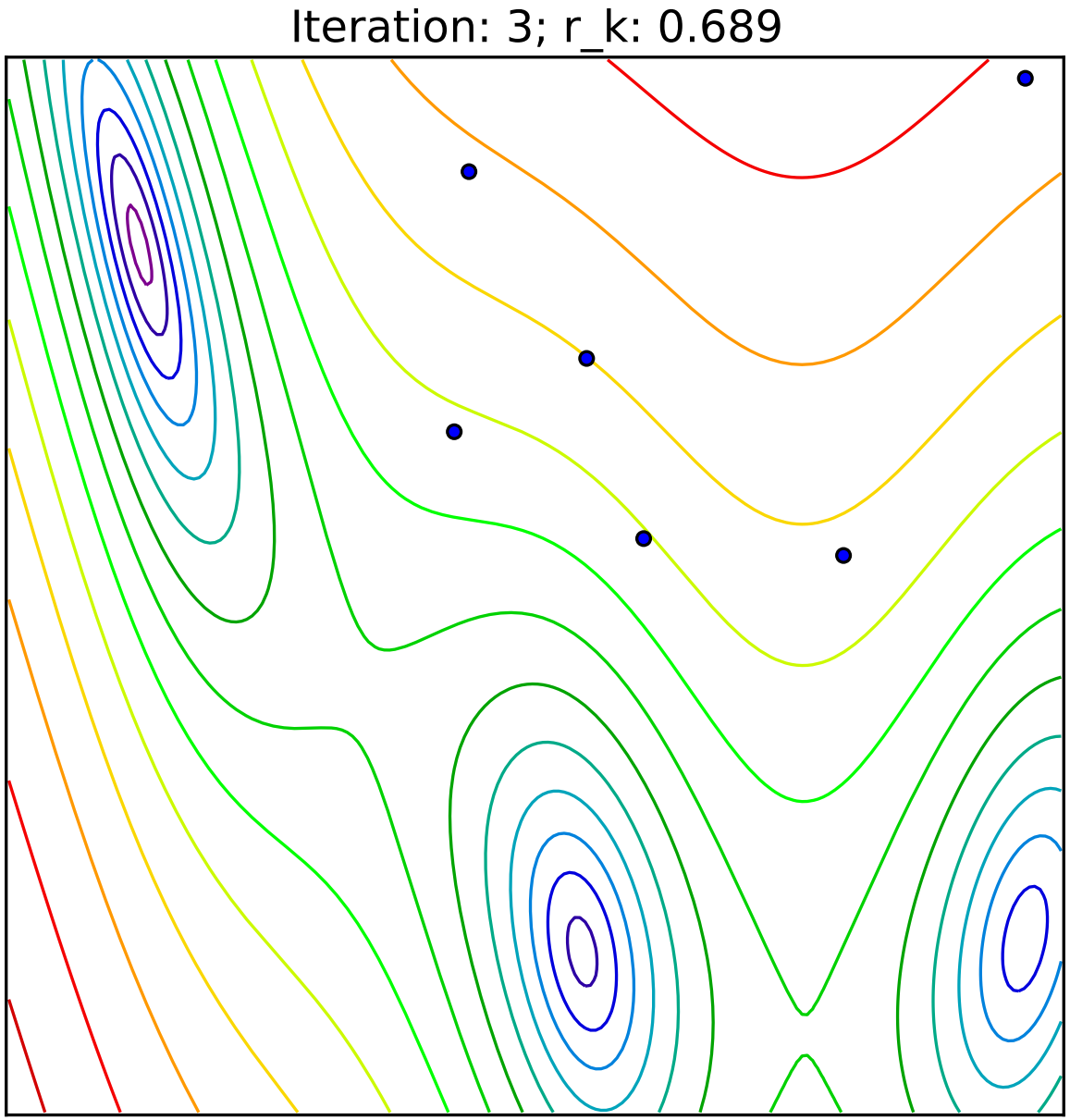
APOSMM



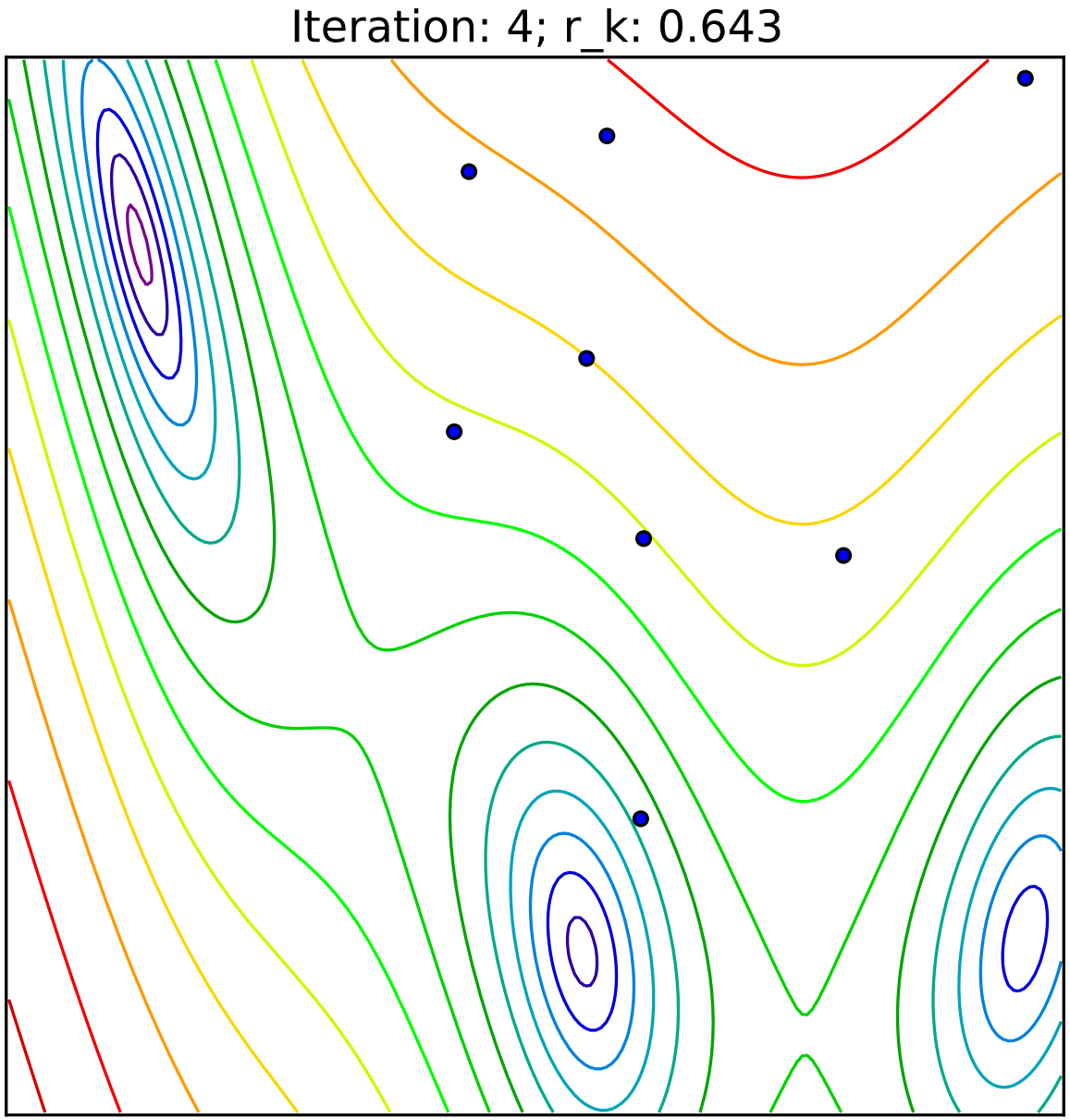
APOSMM



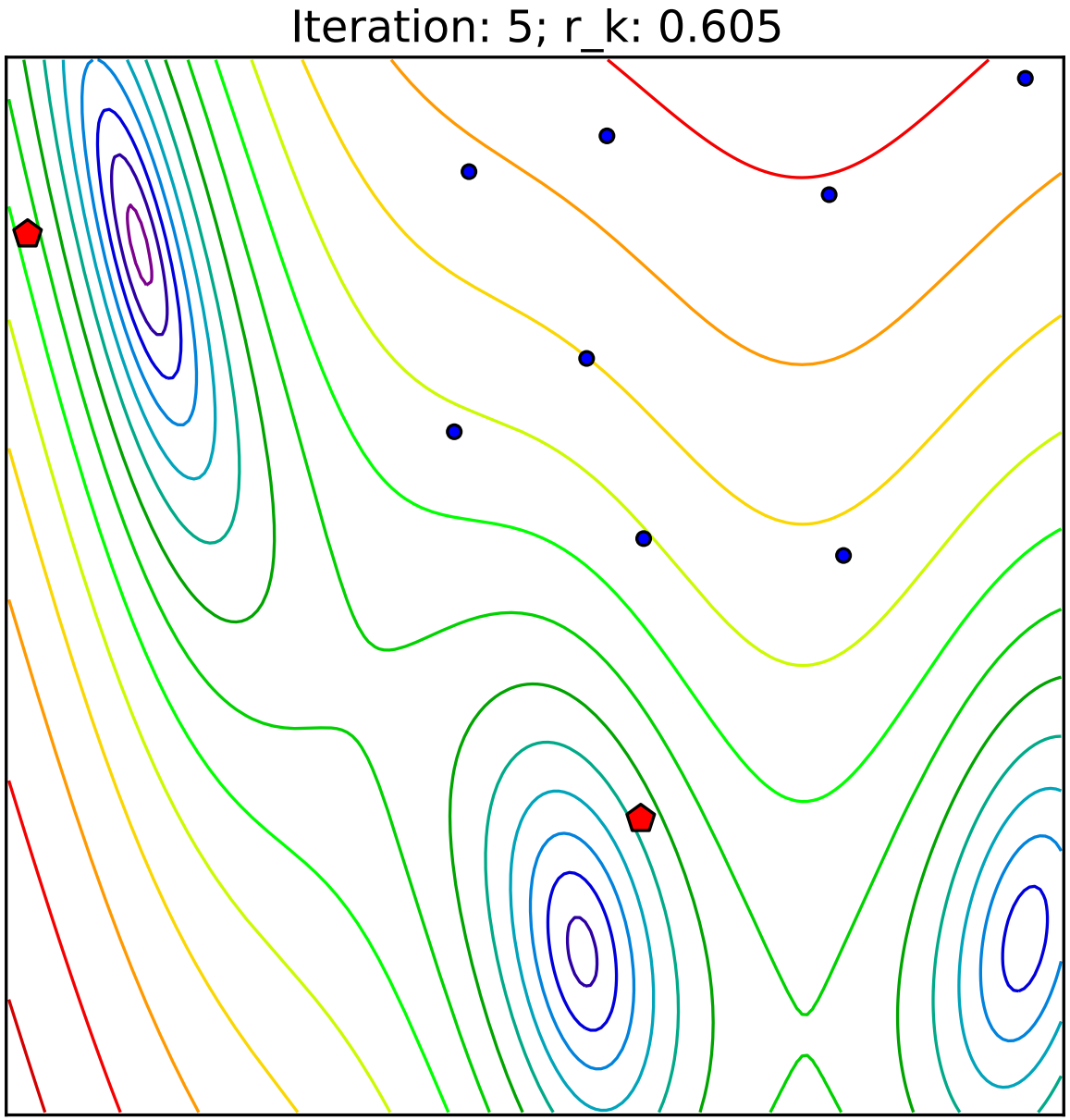
APOSMM



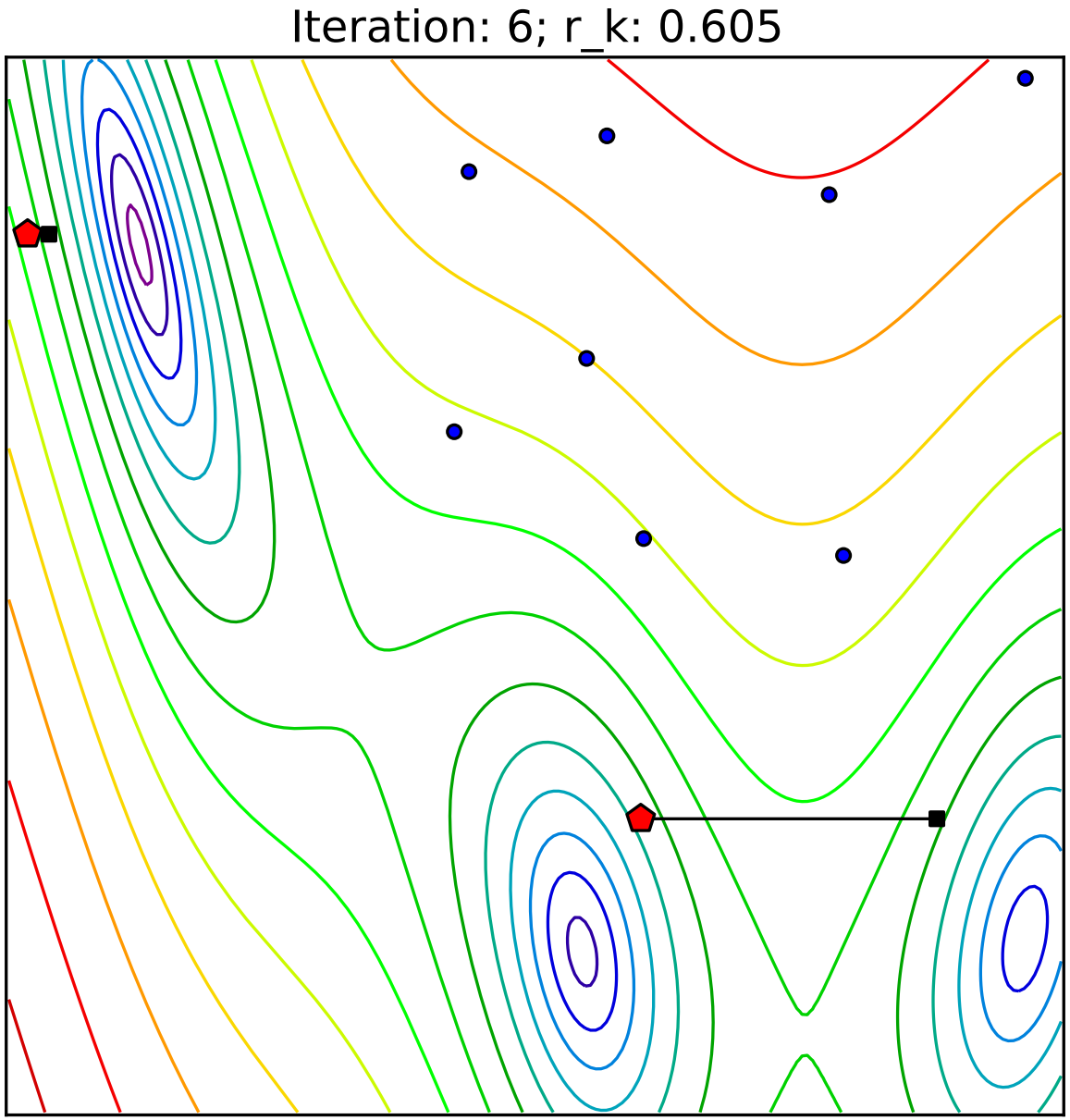
APOSMM



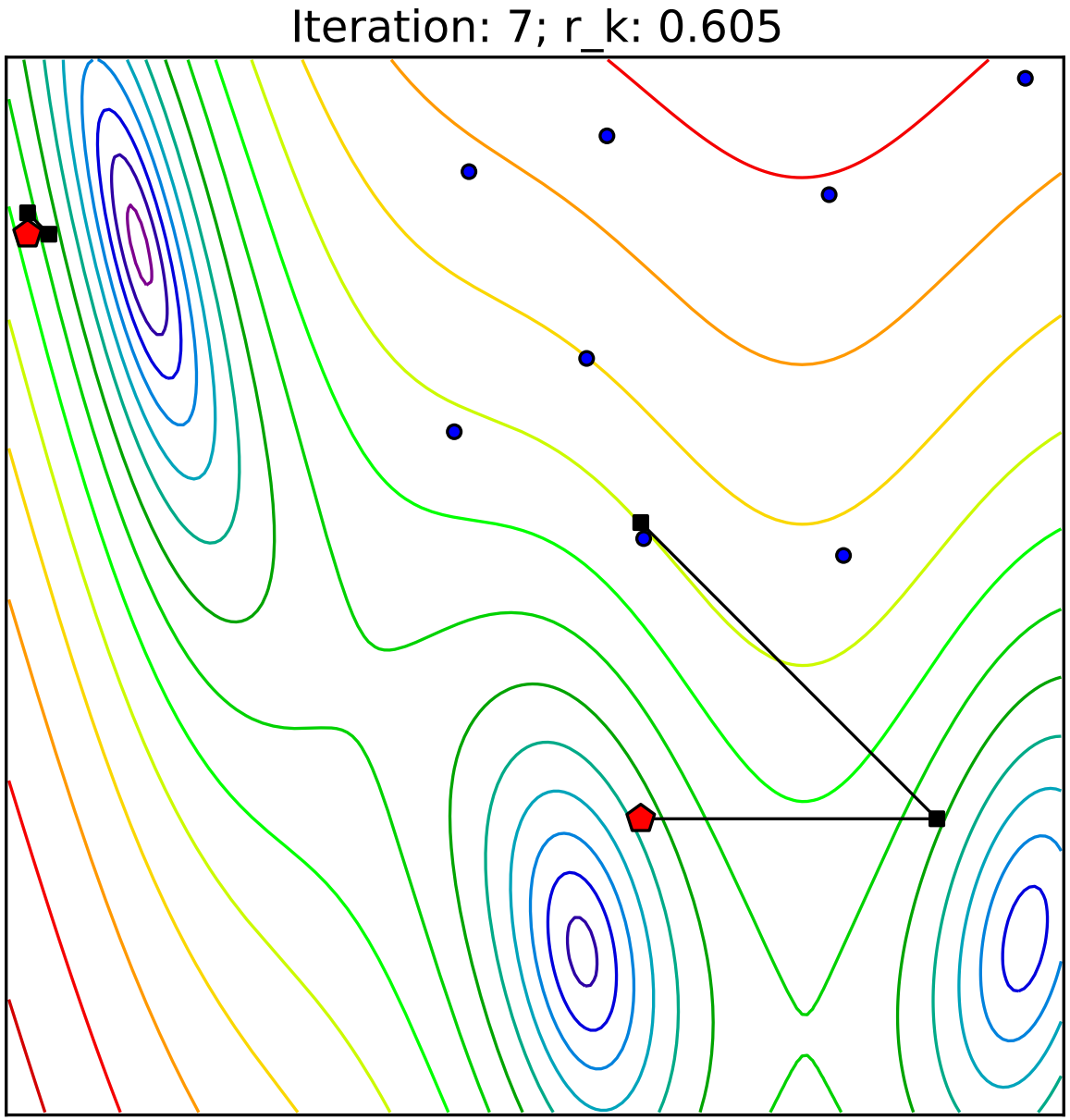
APOSMM



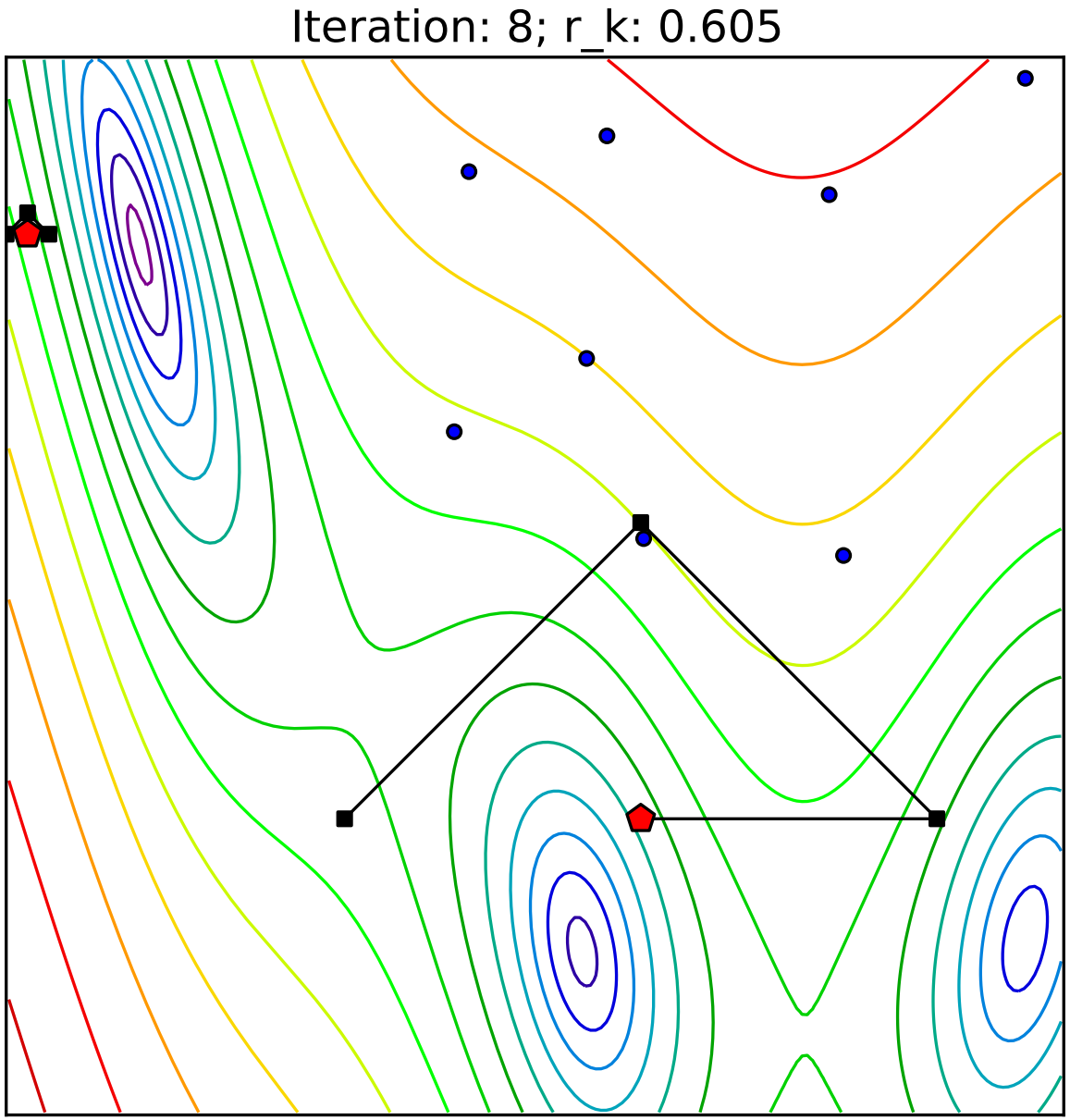
APOSMM



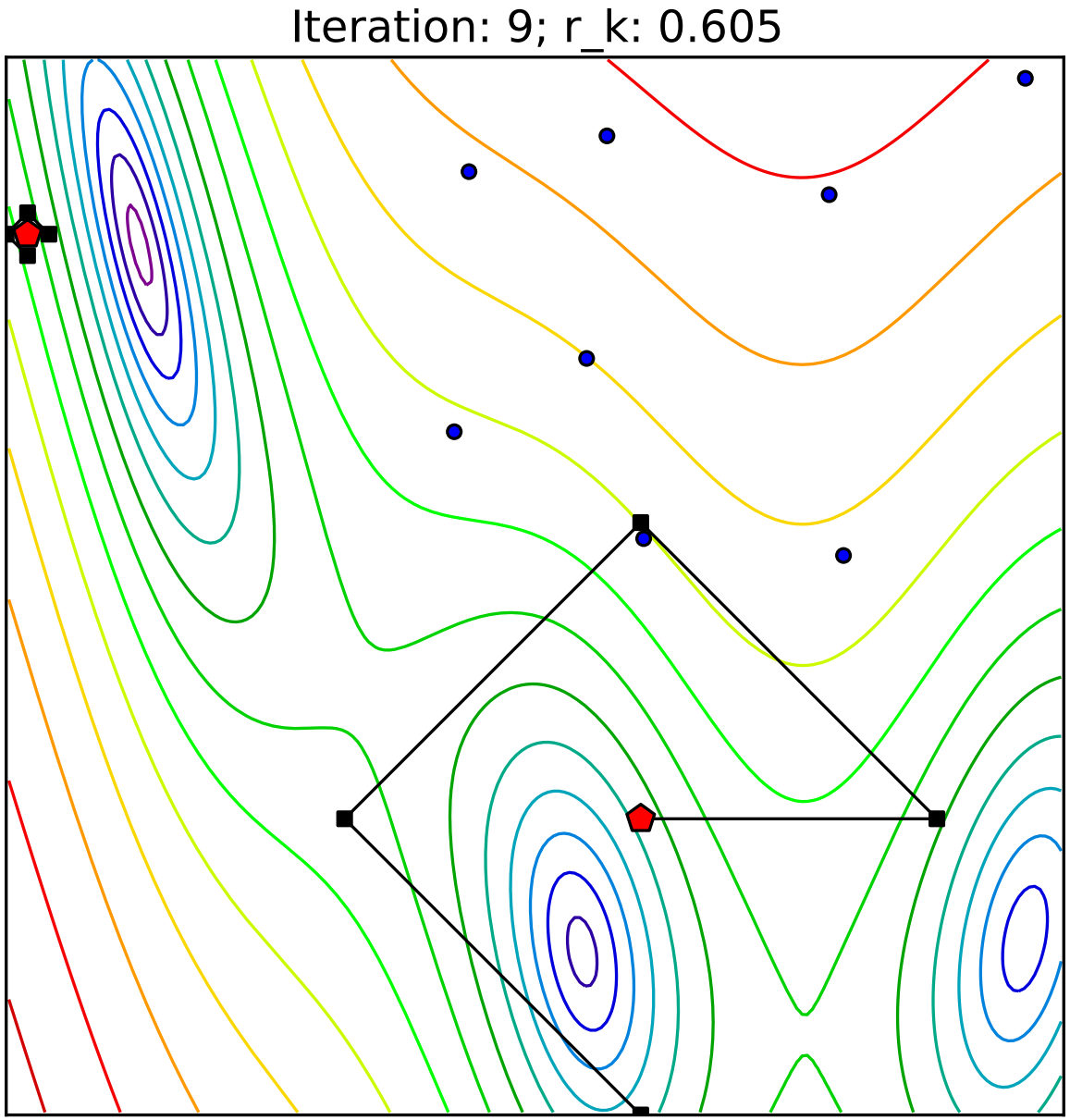
APOSMM



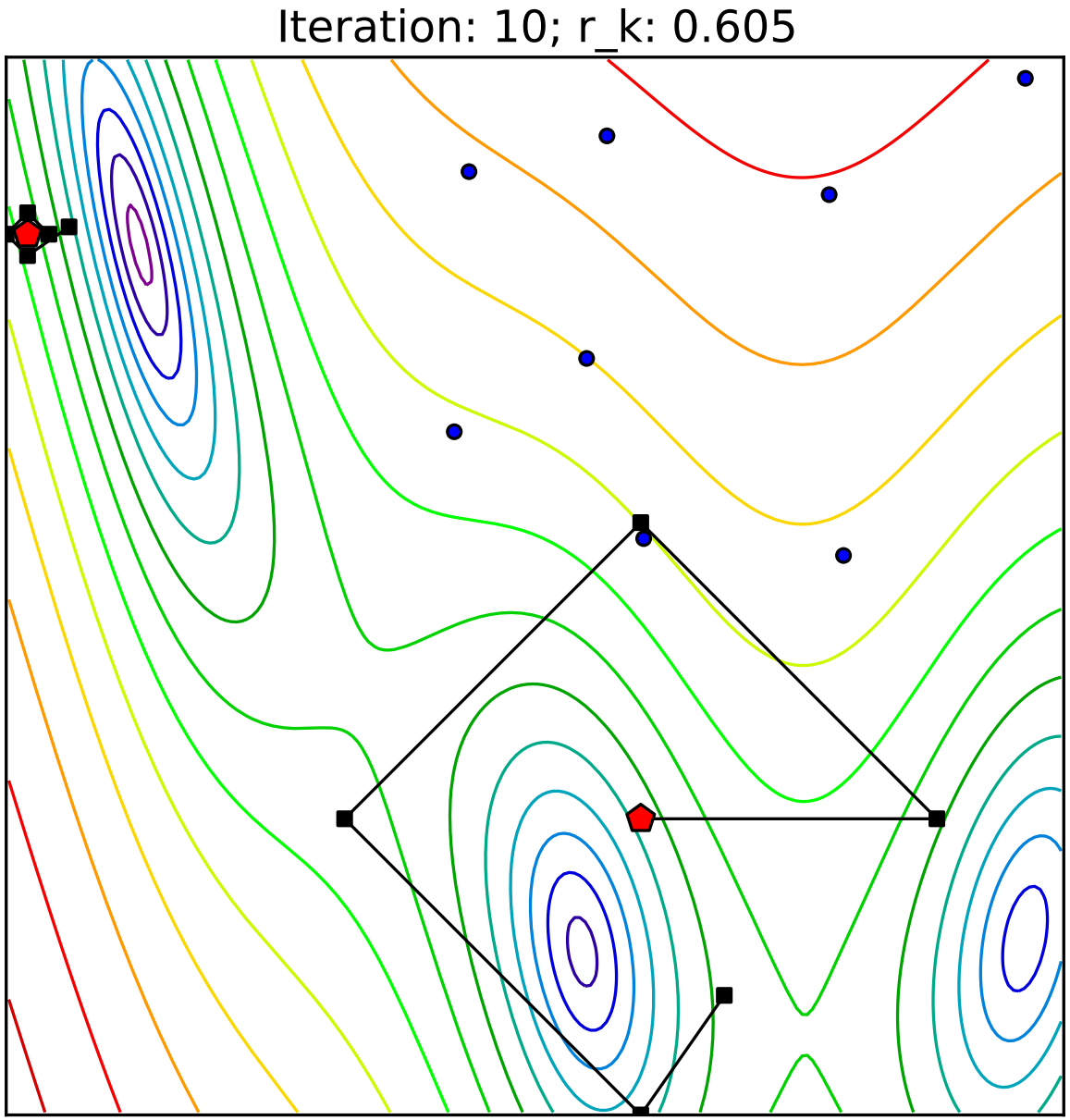
APOSMM



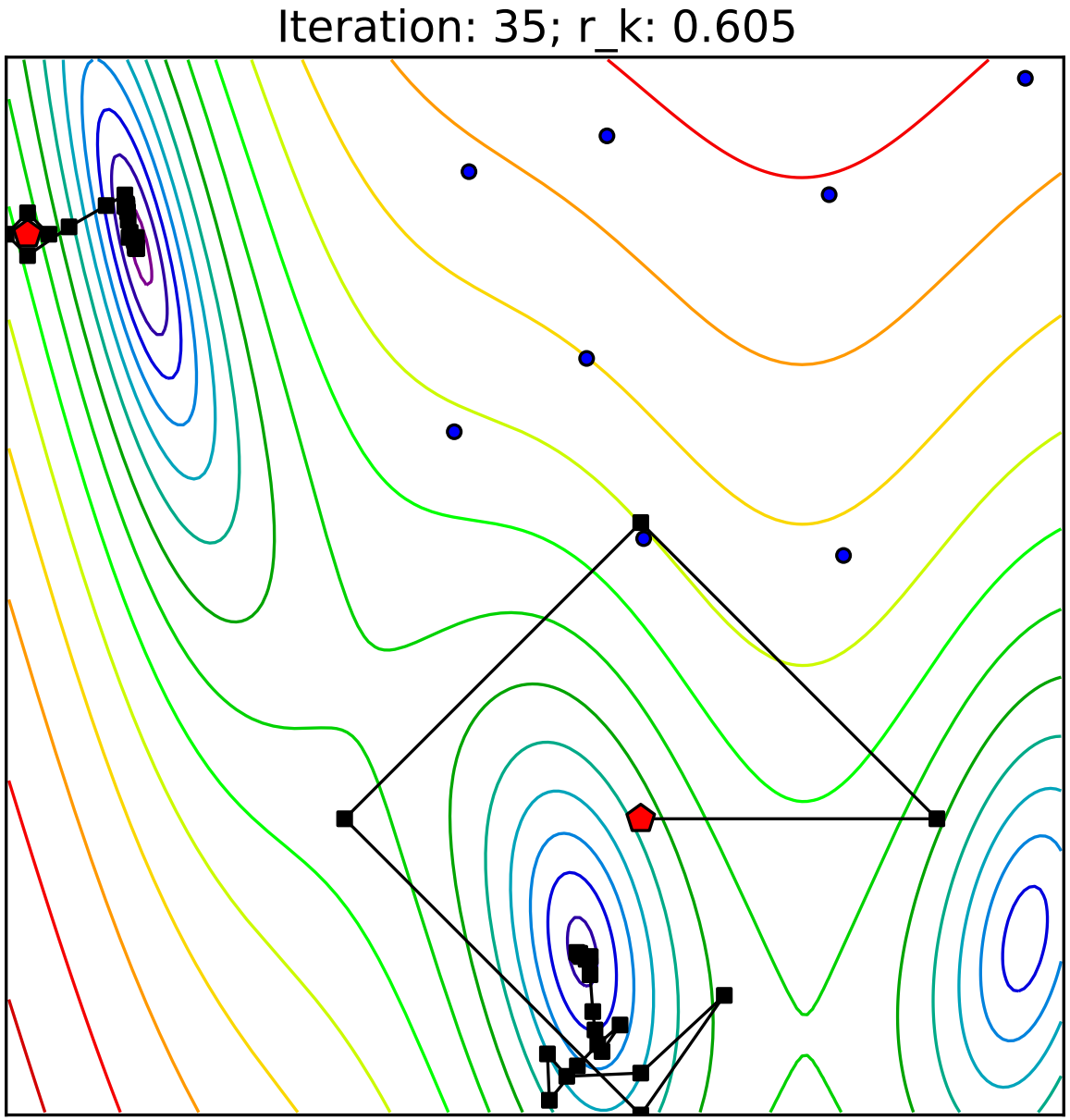
APOSMM



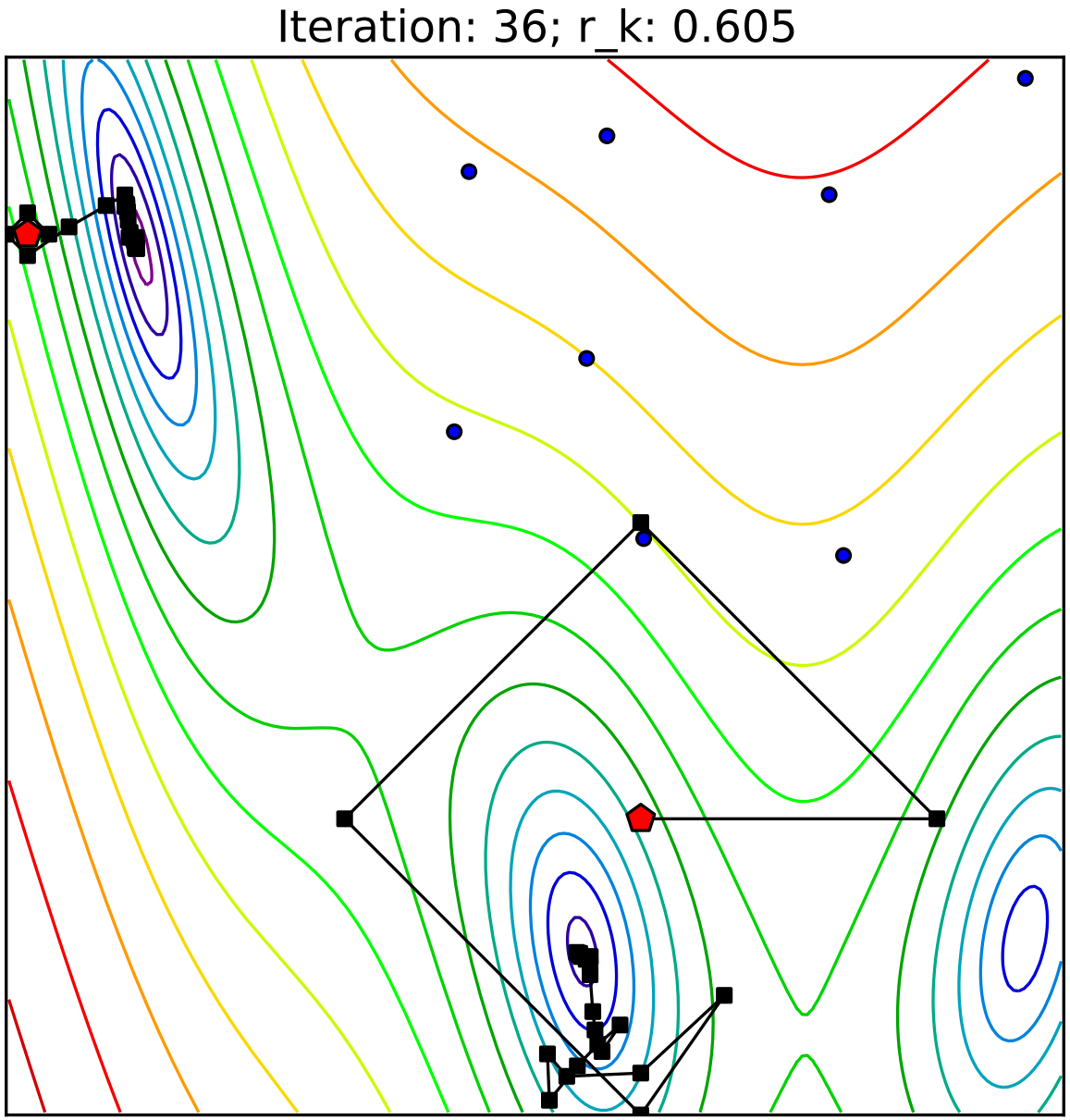
APOSMM



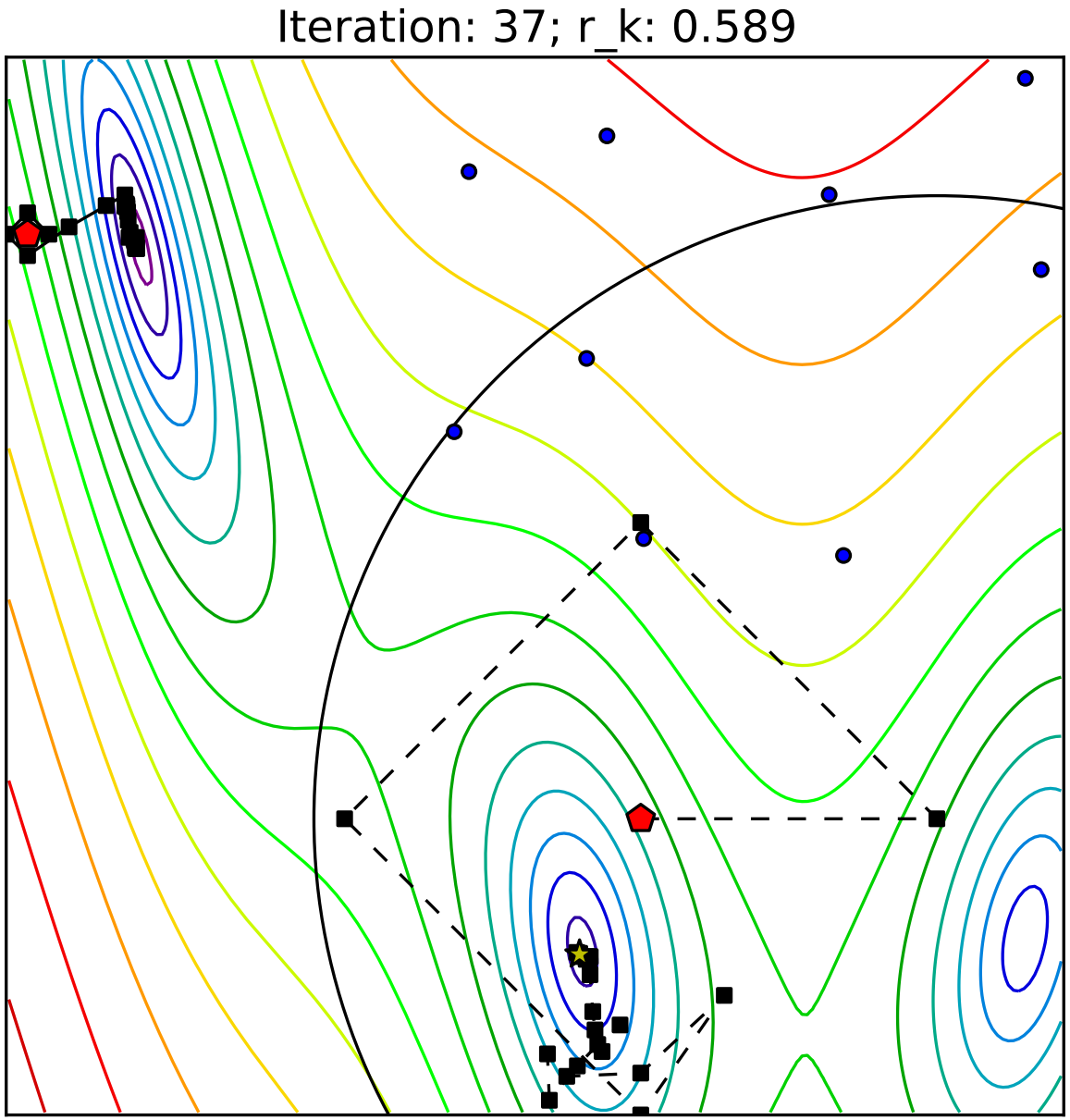
APOSMM



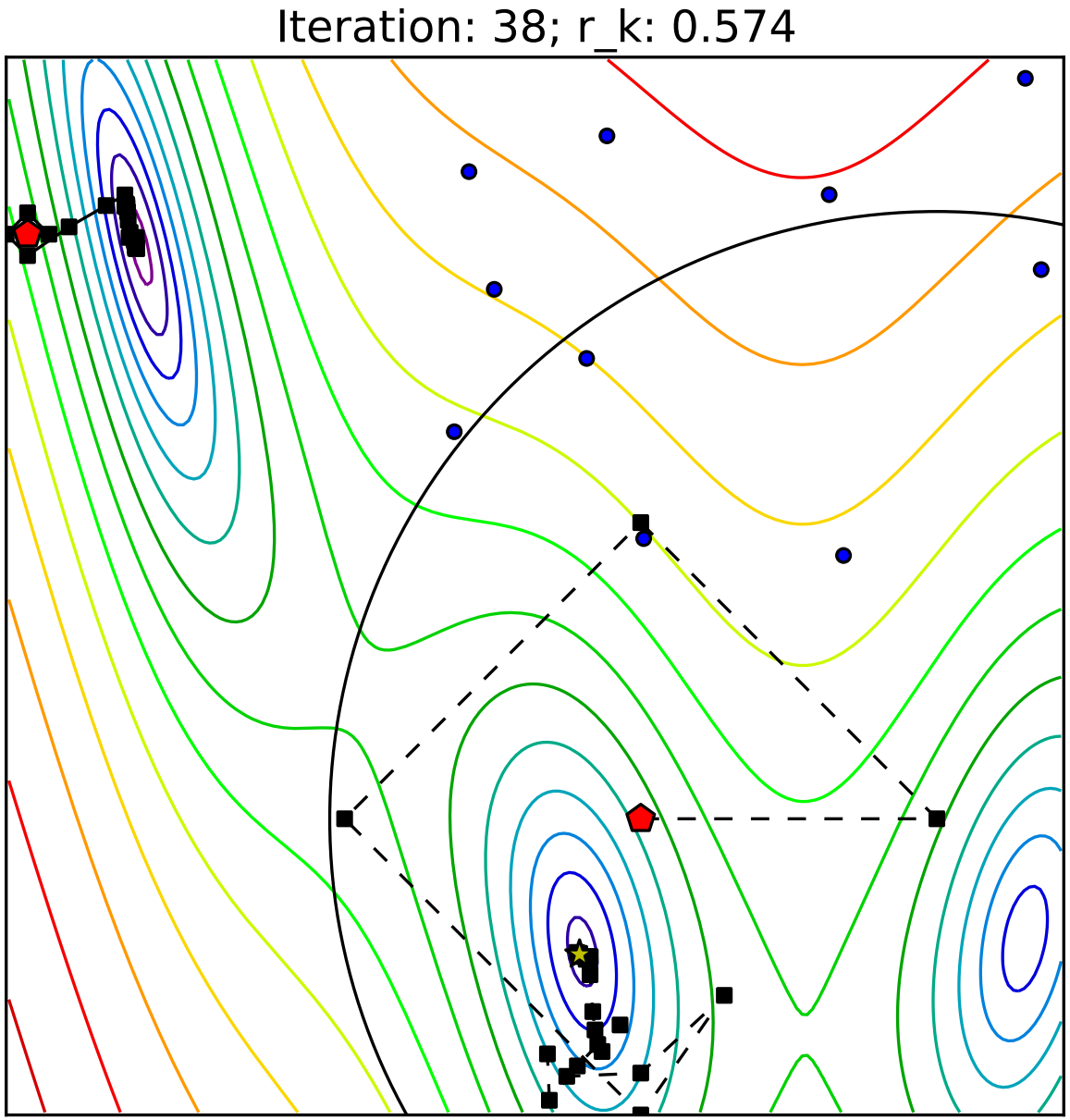
APOSMM



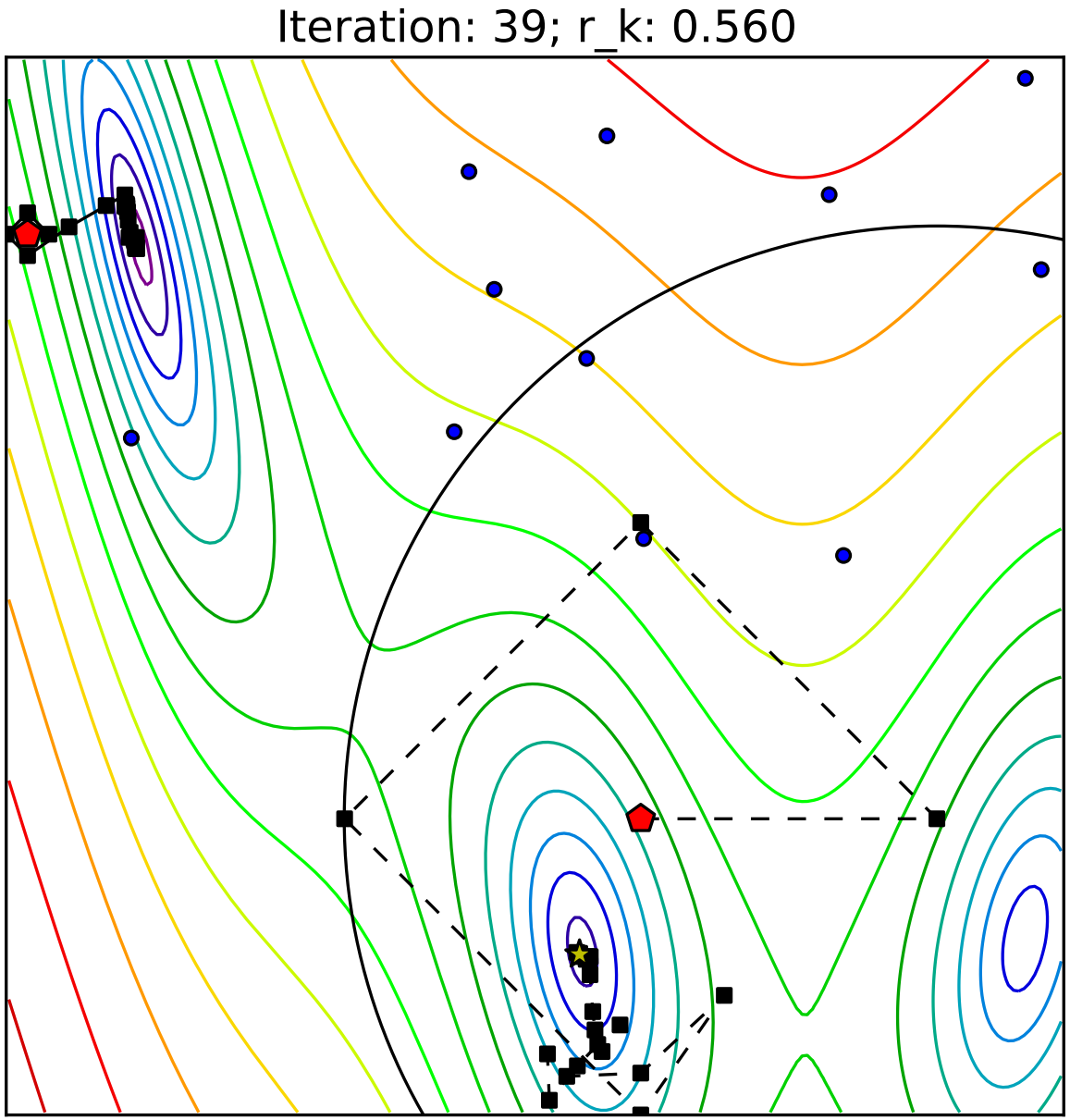
APOSMM



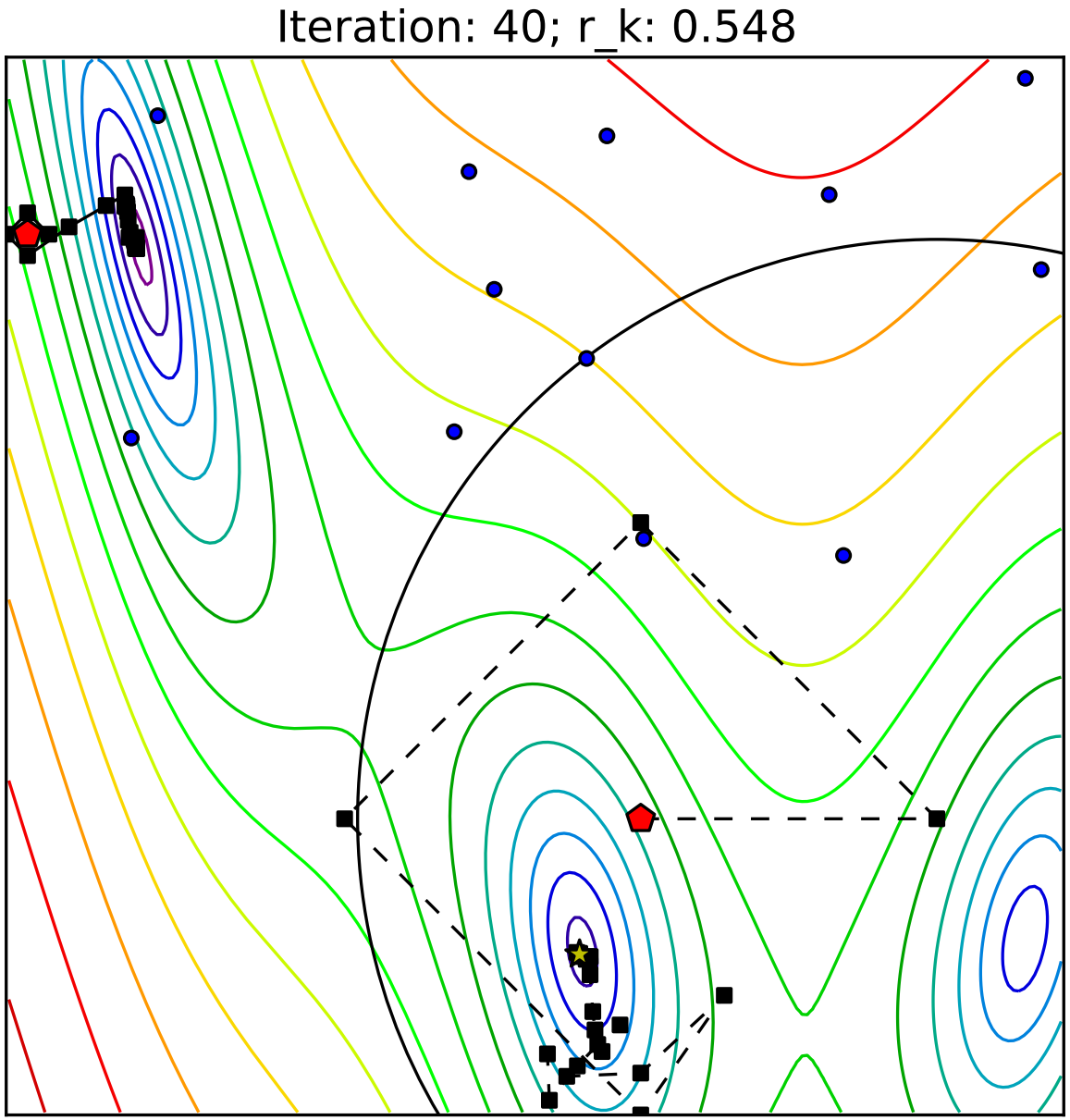
APOSMM



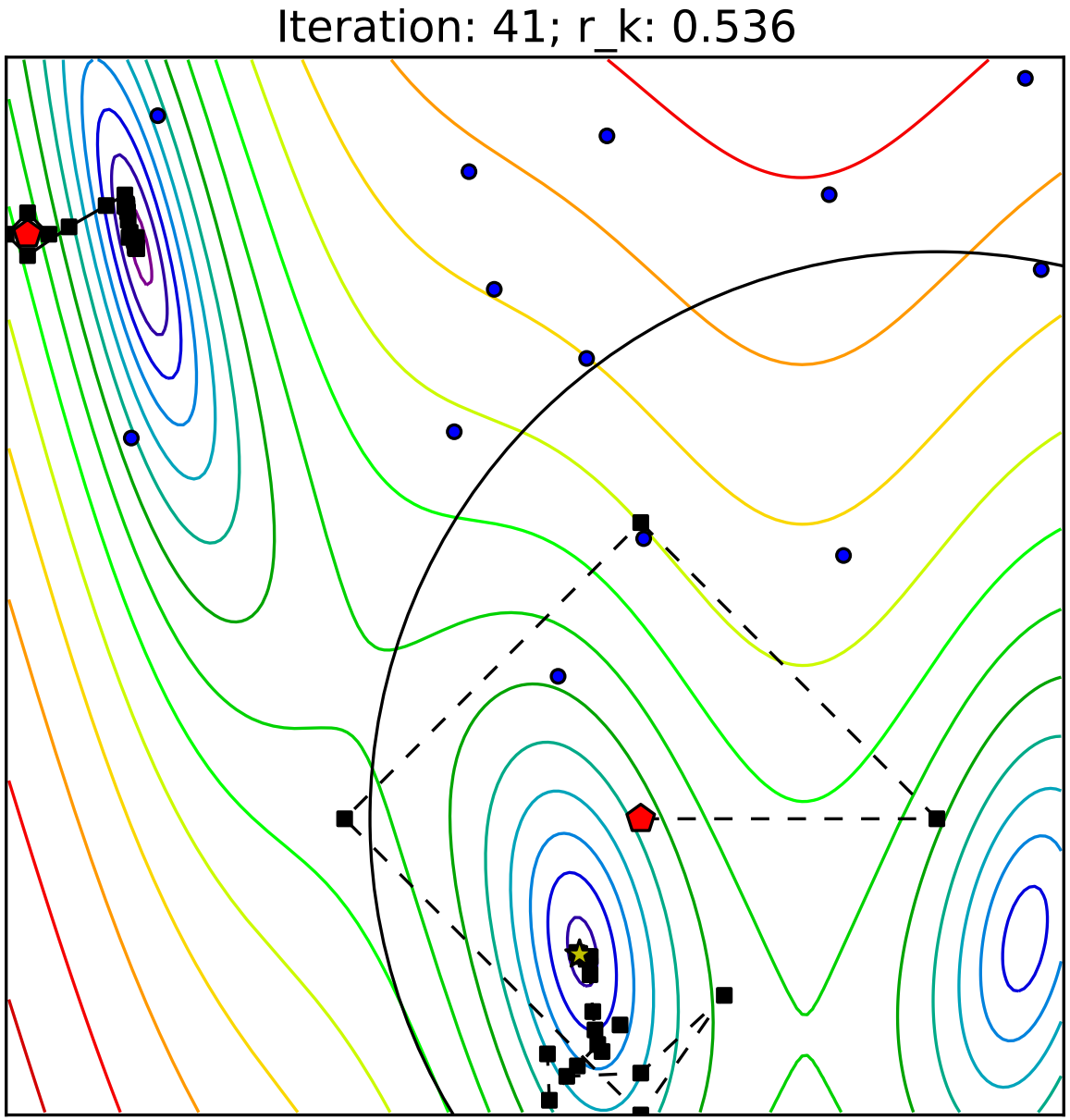
APOSMM



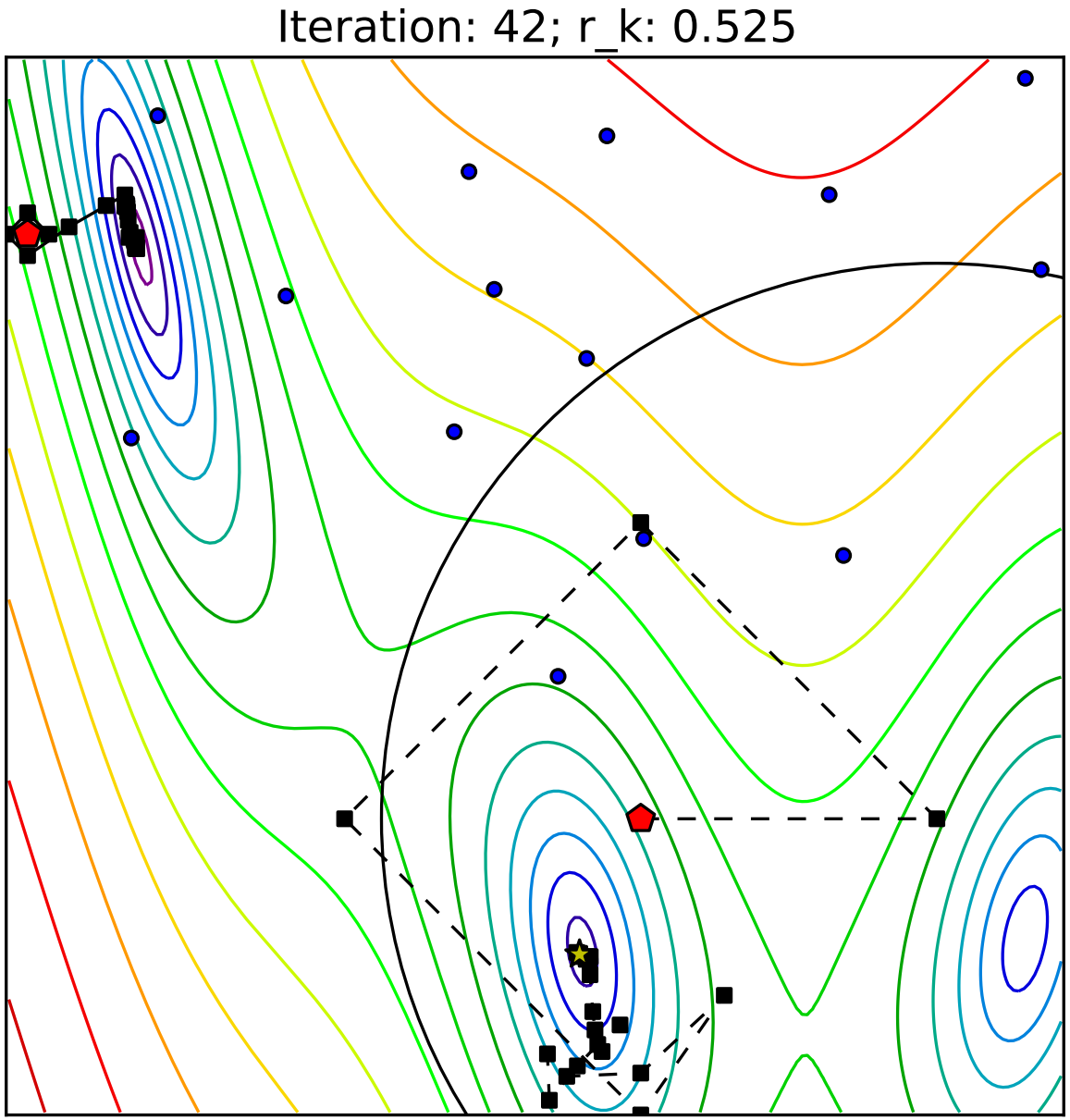
APOSMM



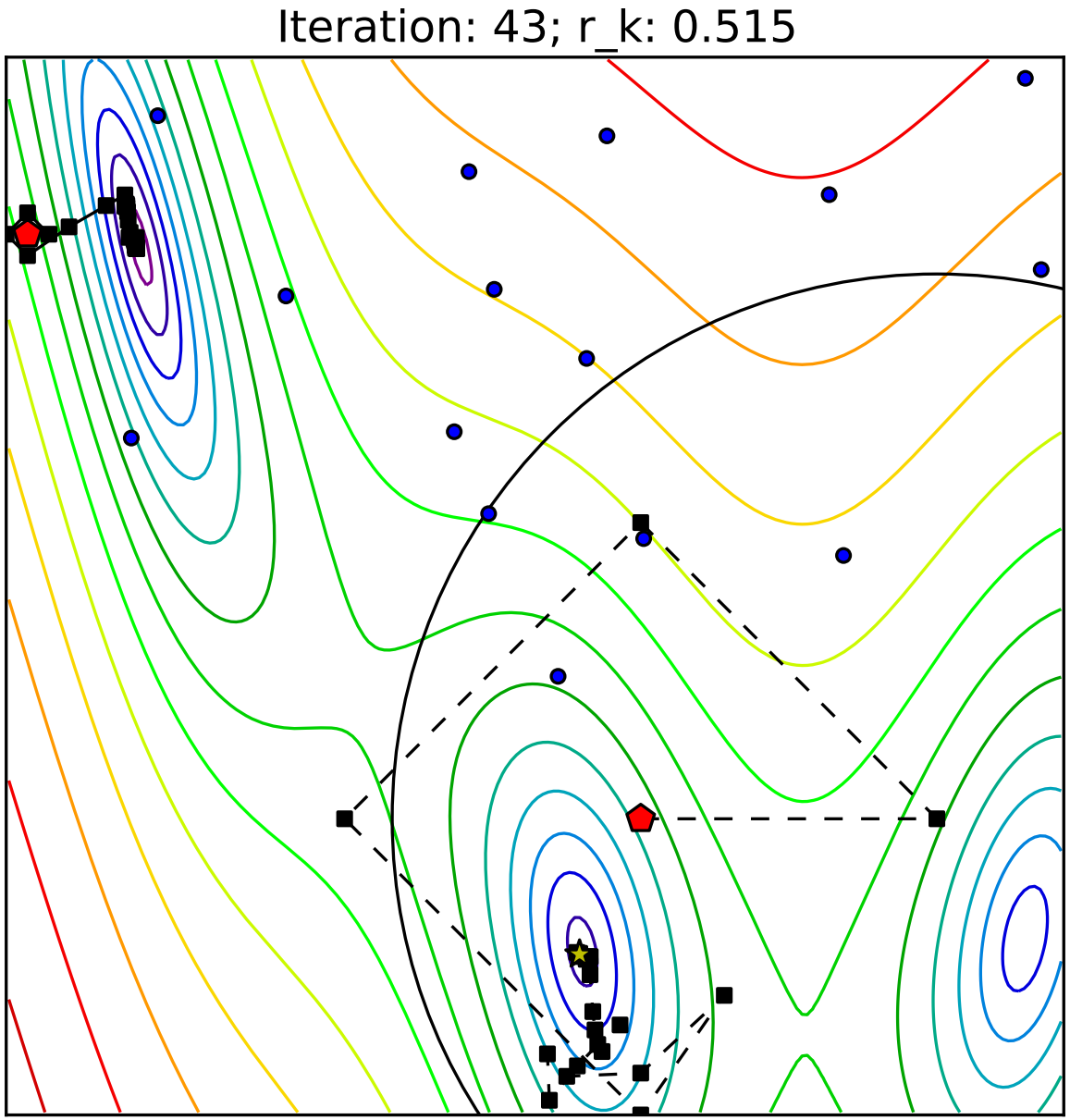
APOSMM



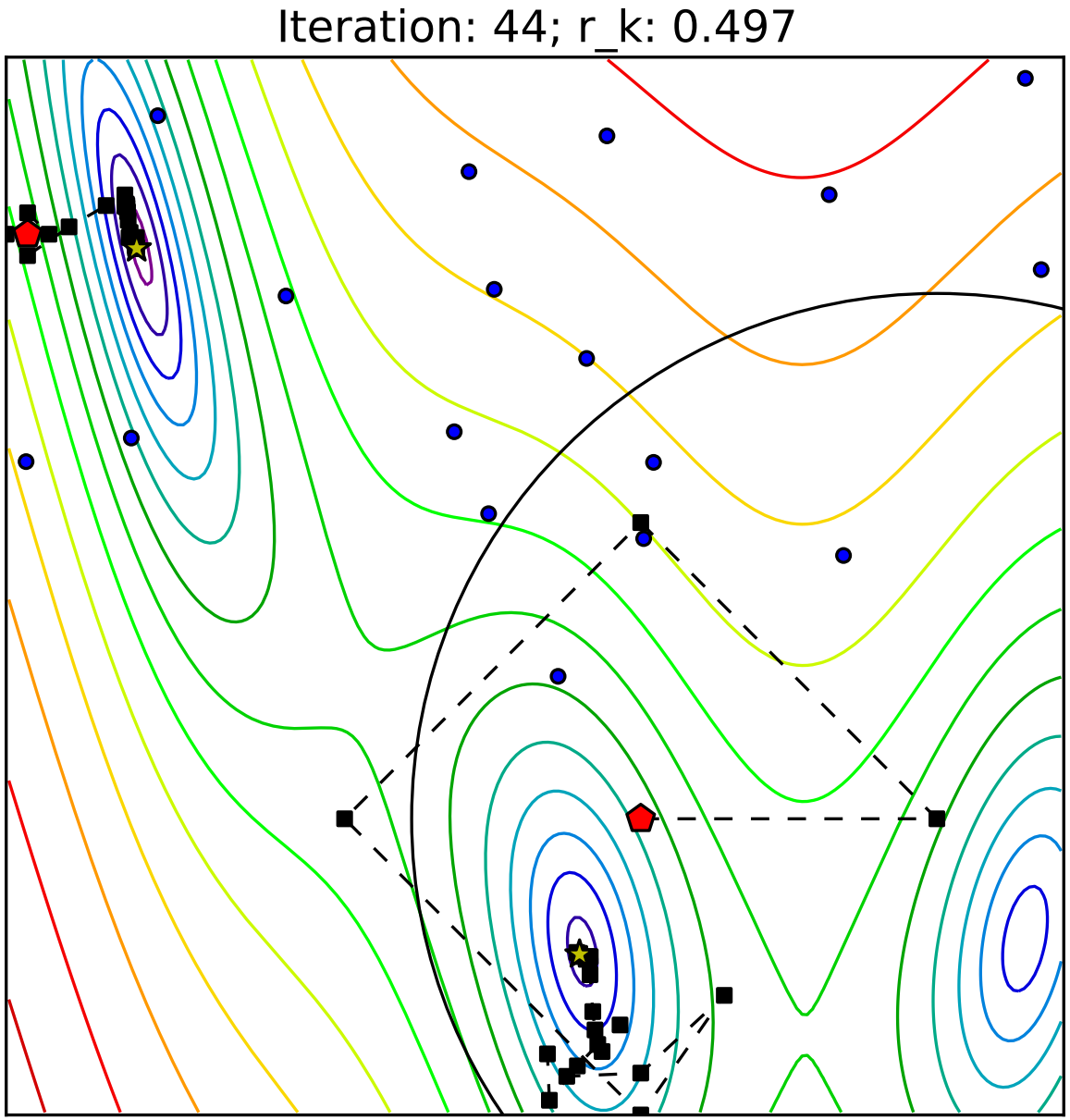
APOSMM



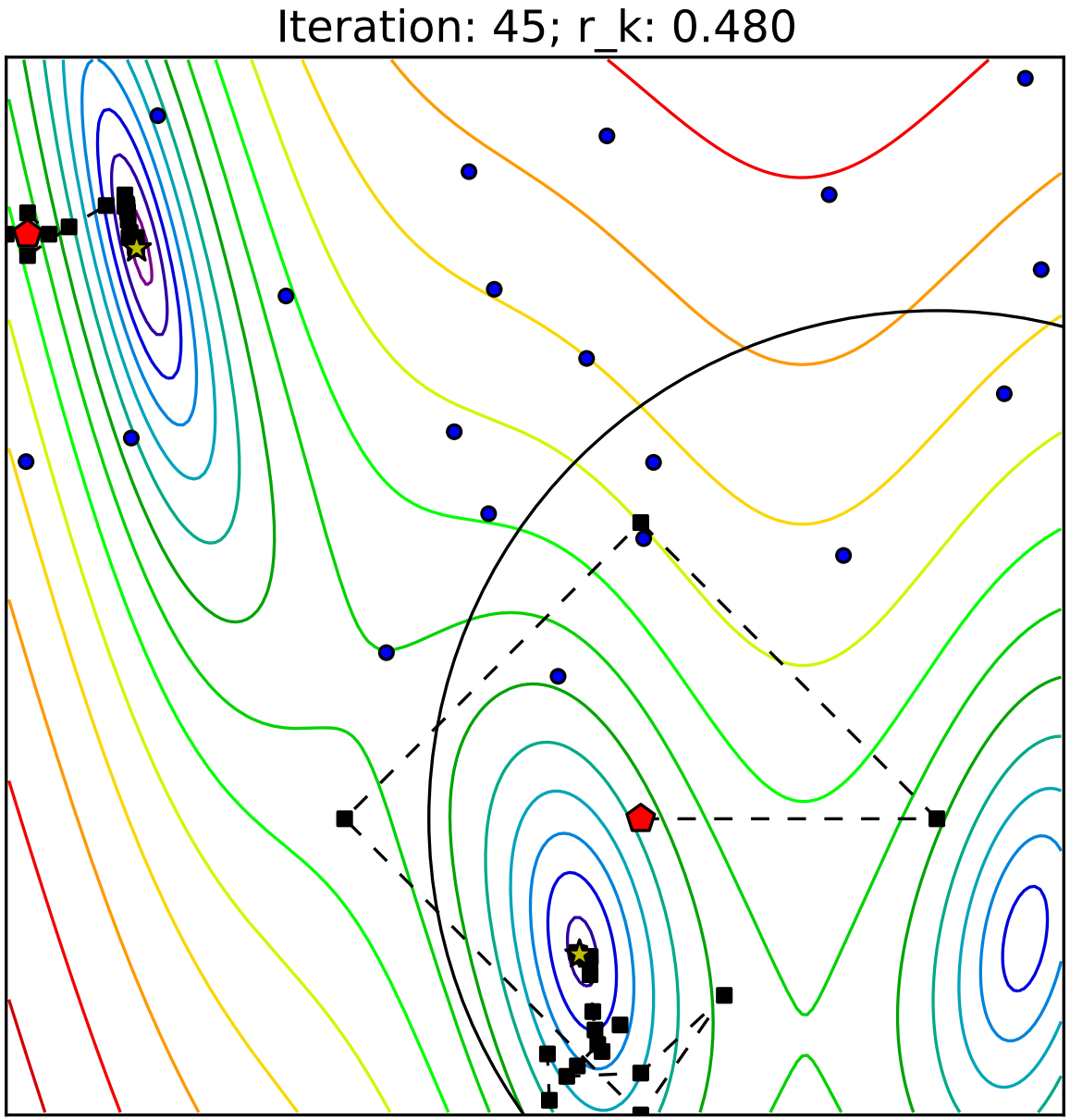
APOSMM



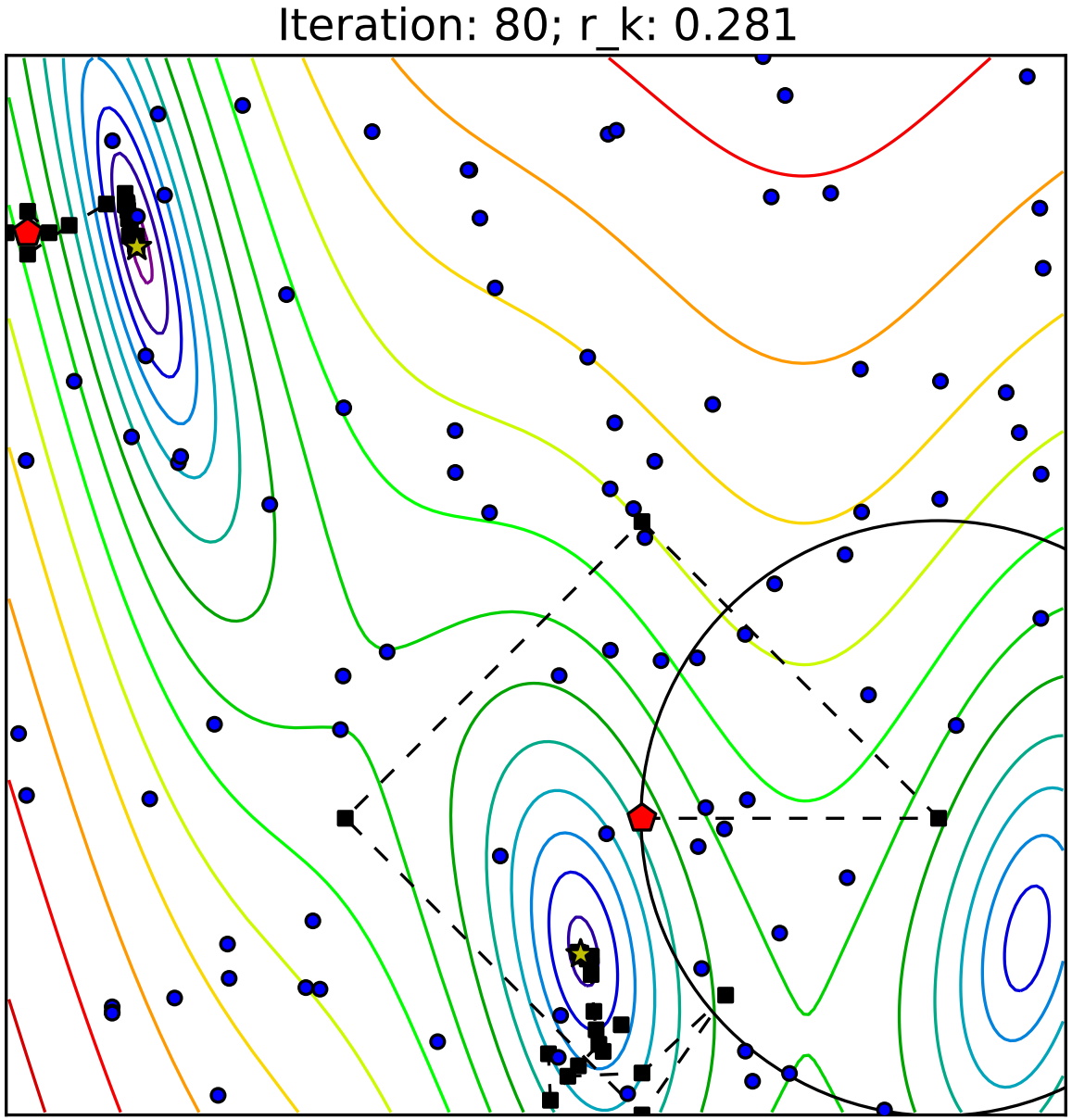
APOSMM



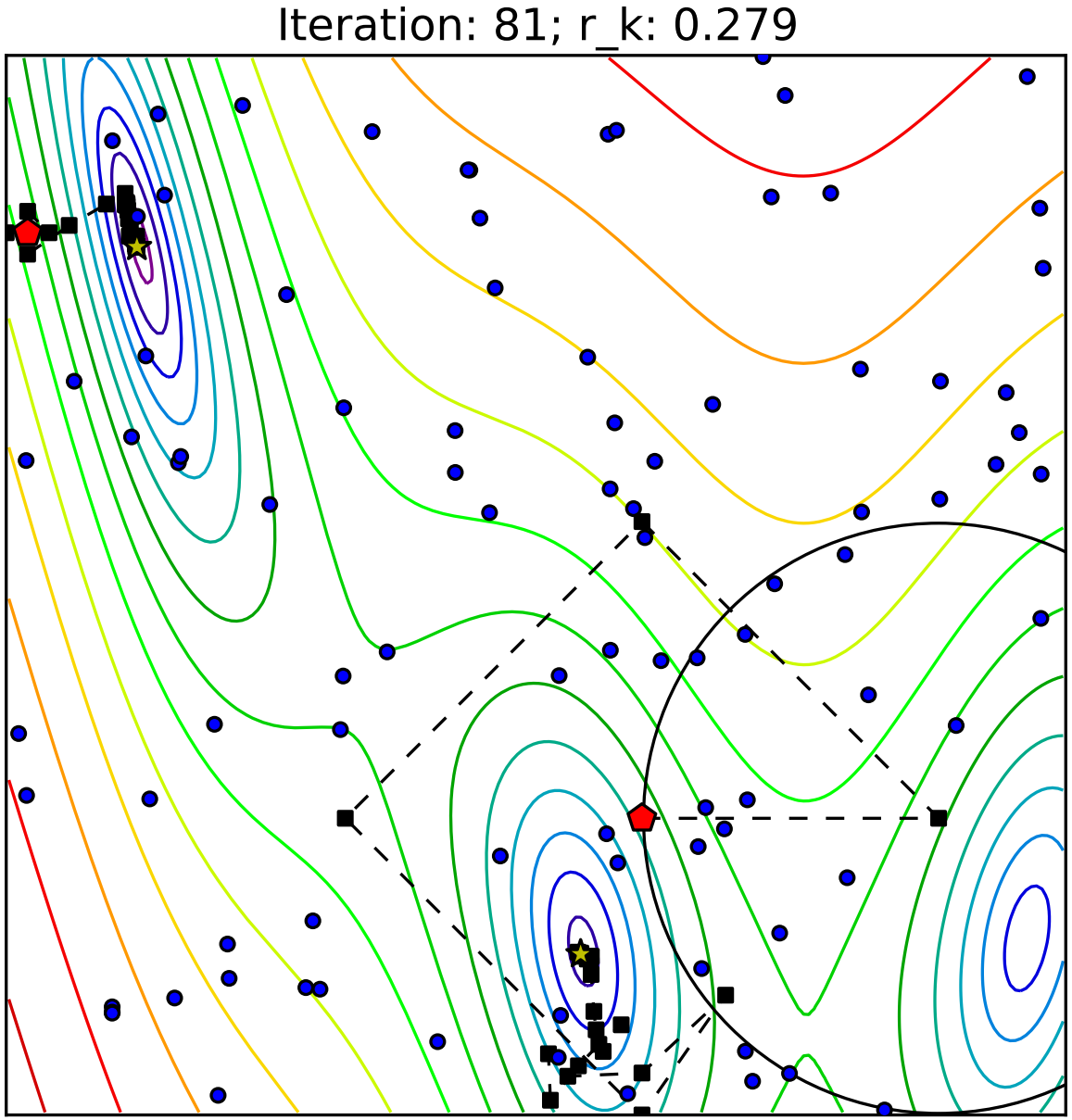
APOSMM



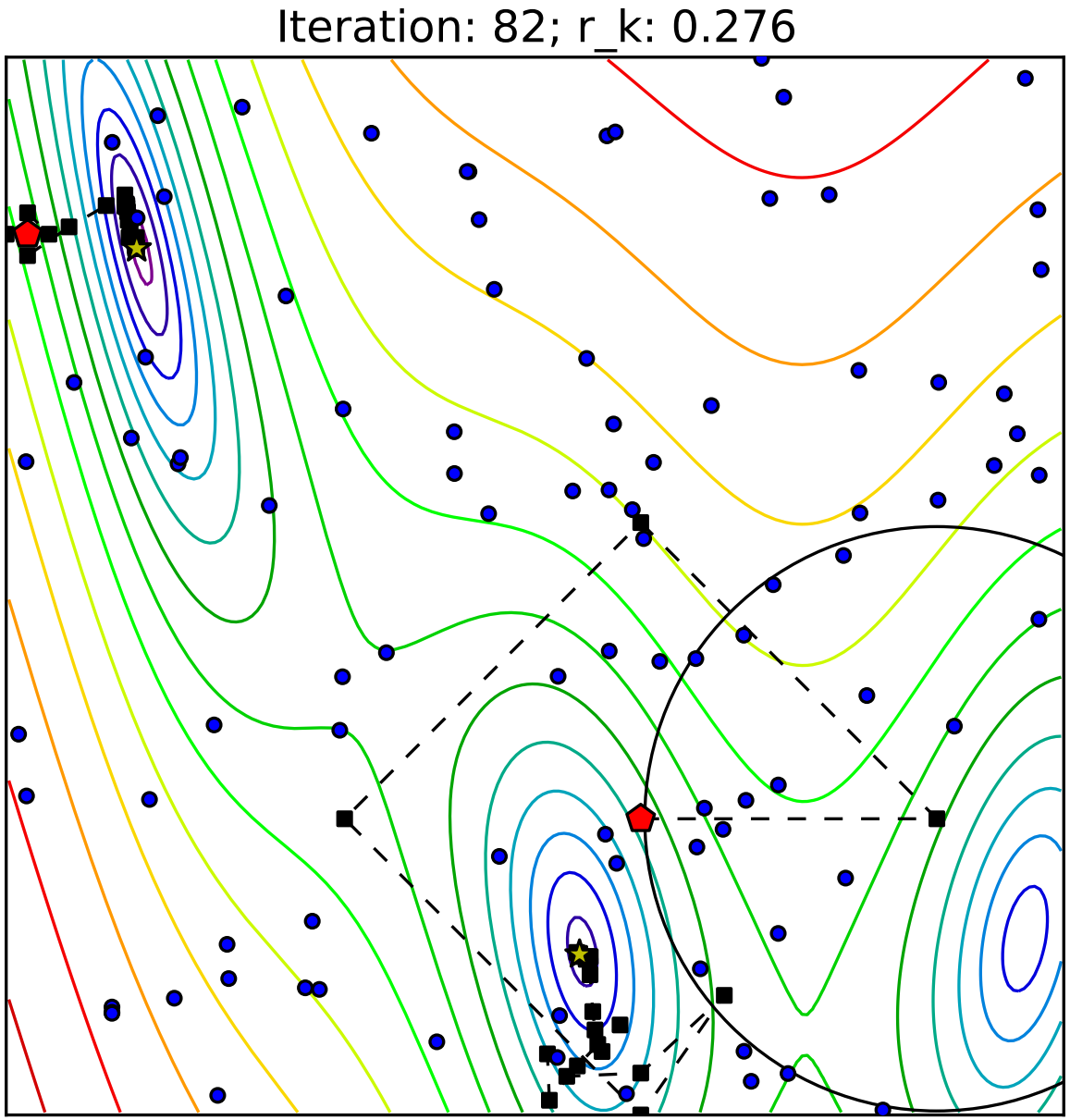
APOSMM



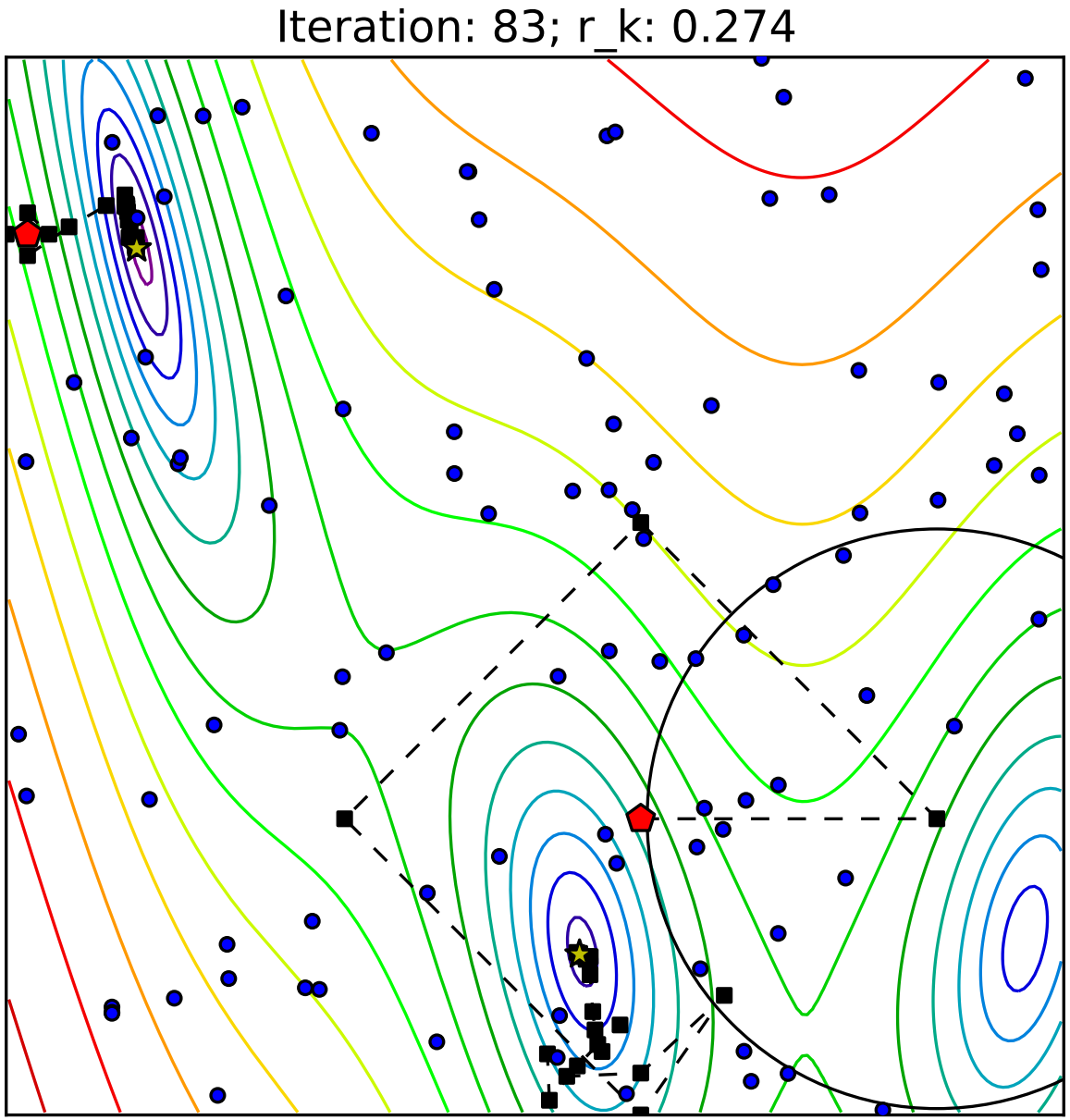
APOSMM



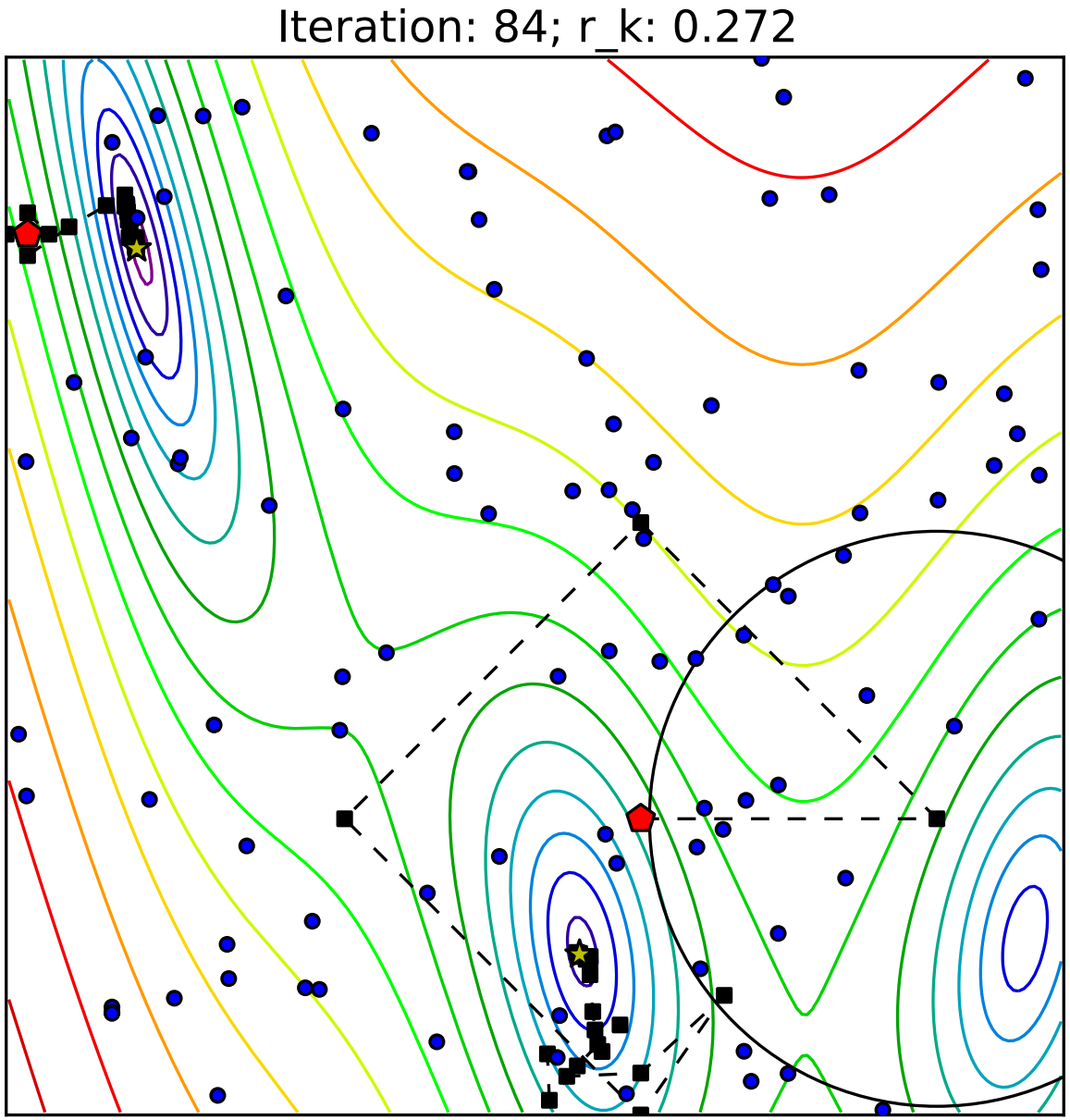
APOSMM



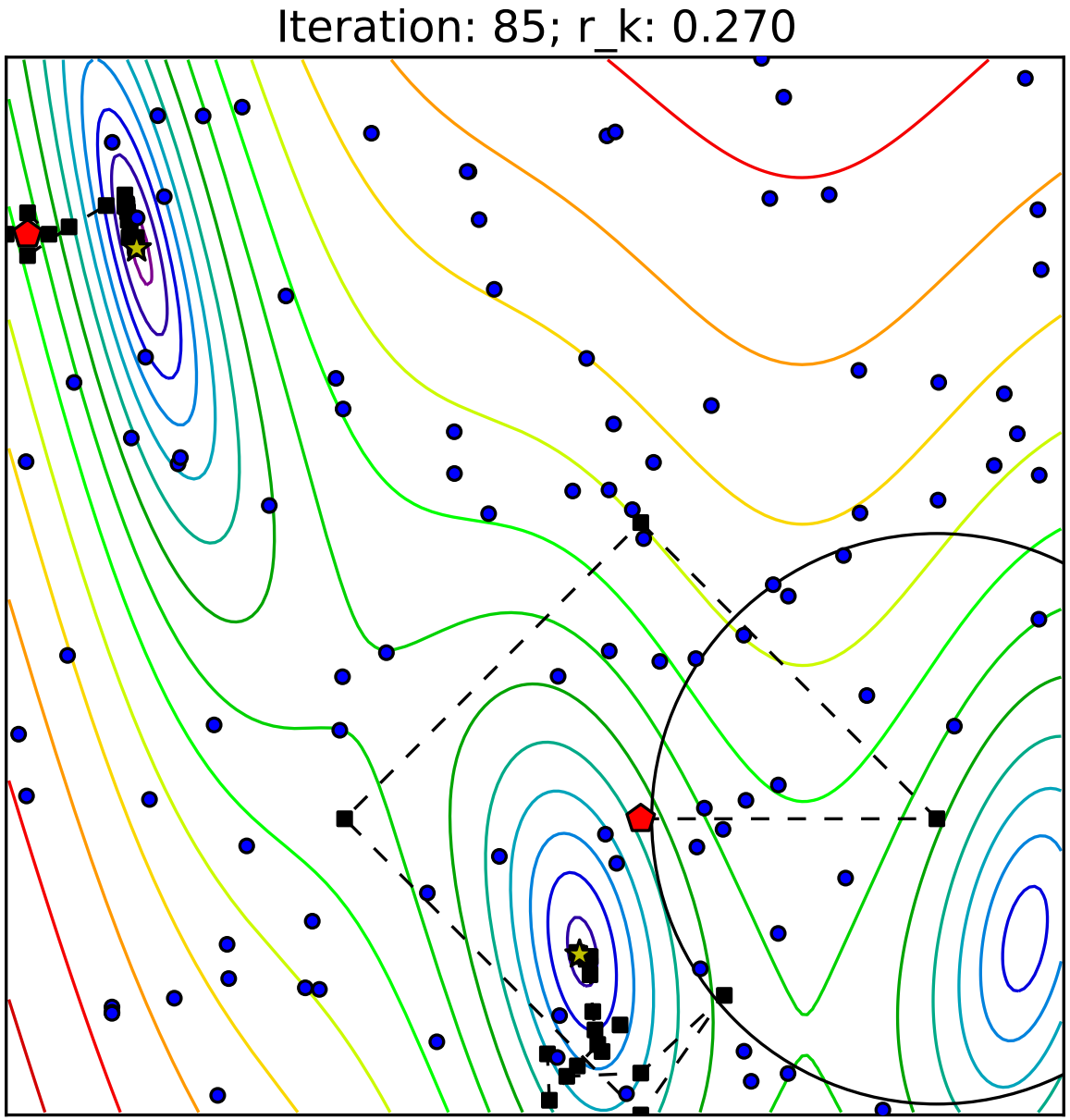
APOSMM



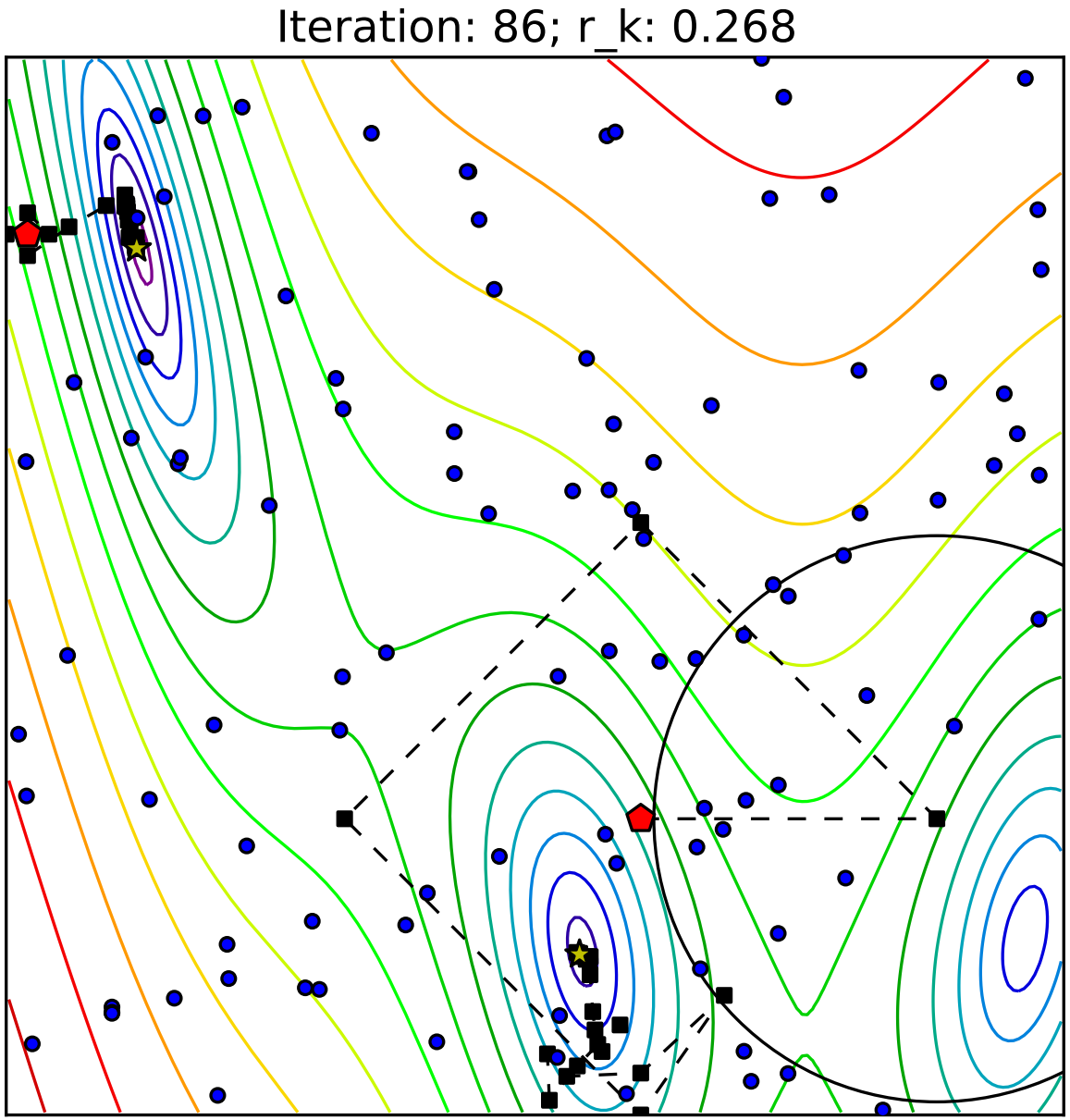
APOSMM



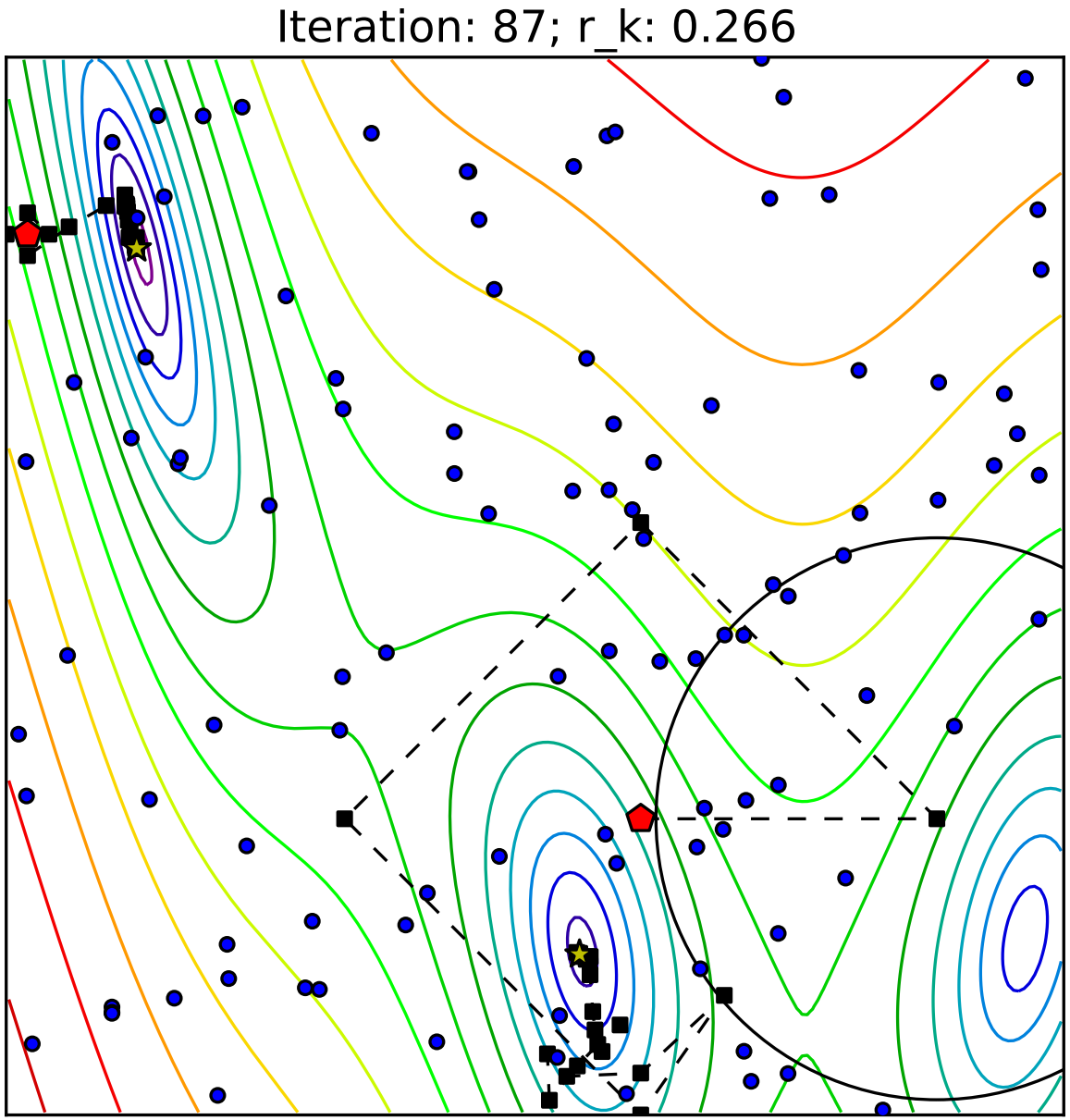
APOSMM



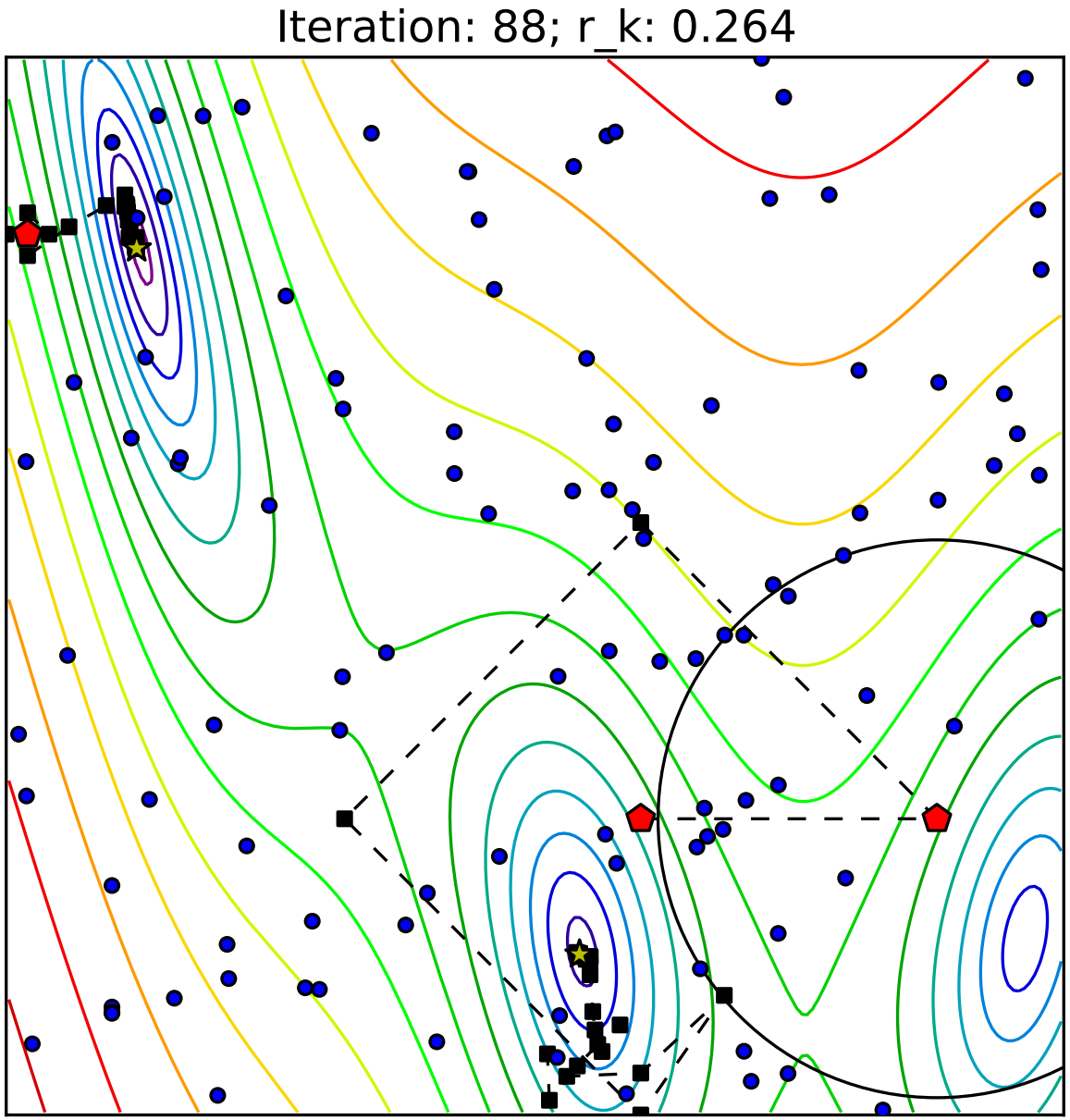
APOSMM



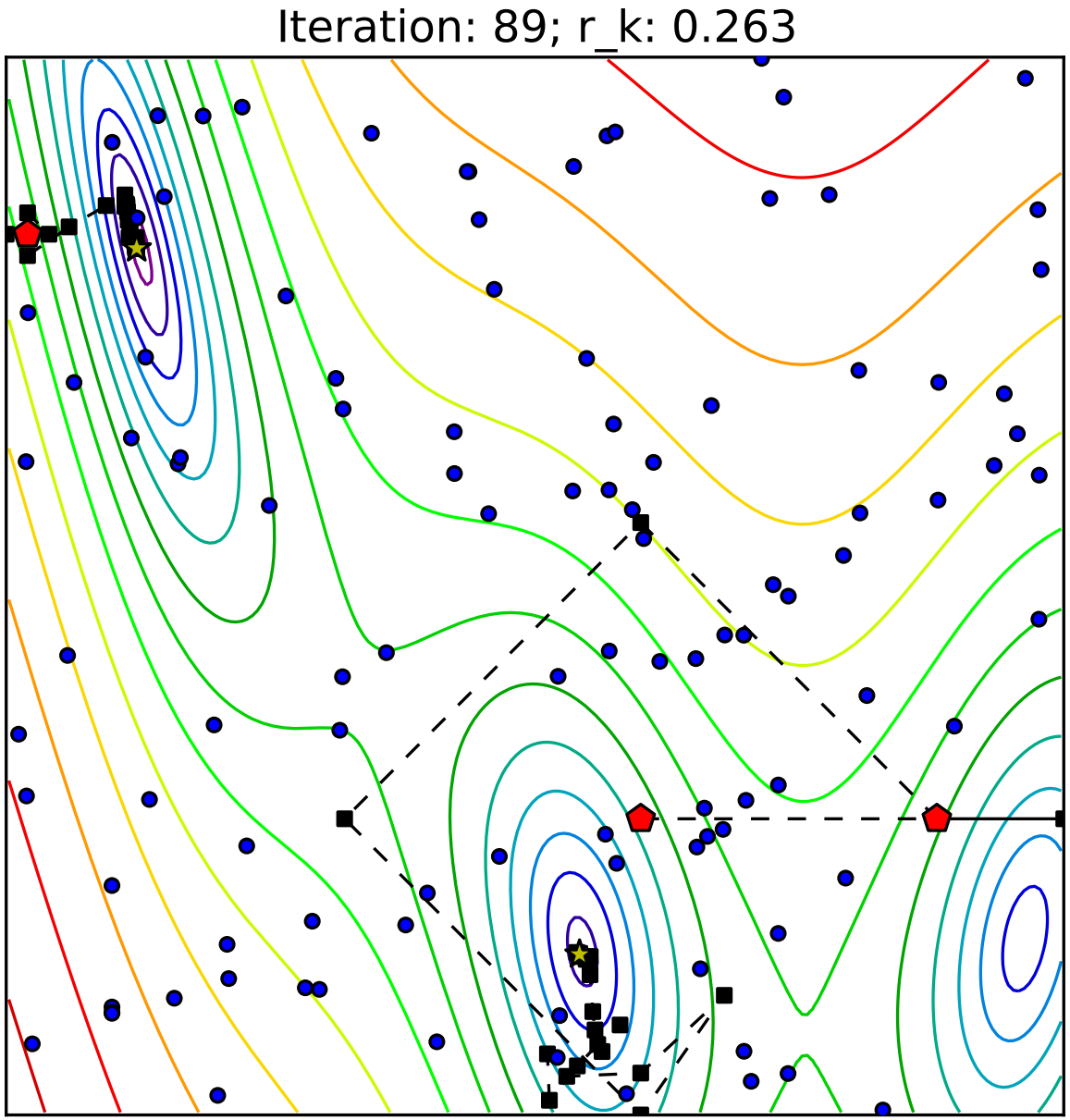
APOSMM



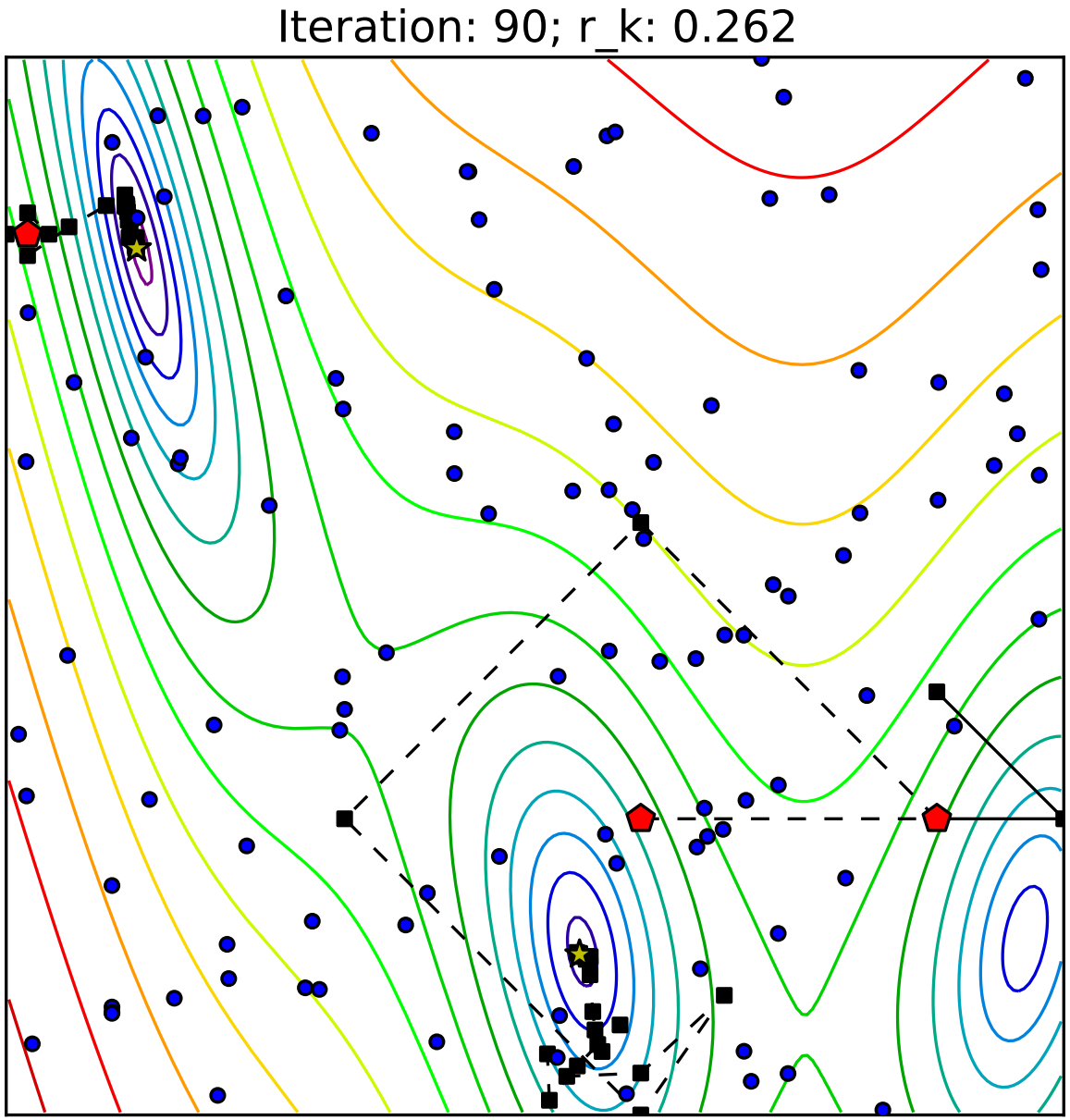
APOSMM



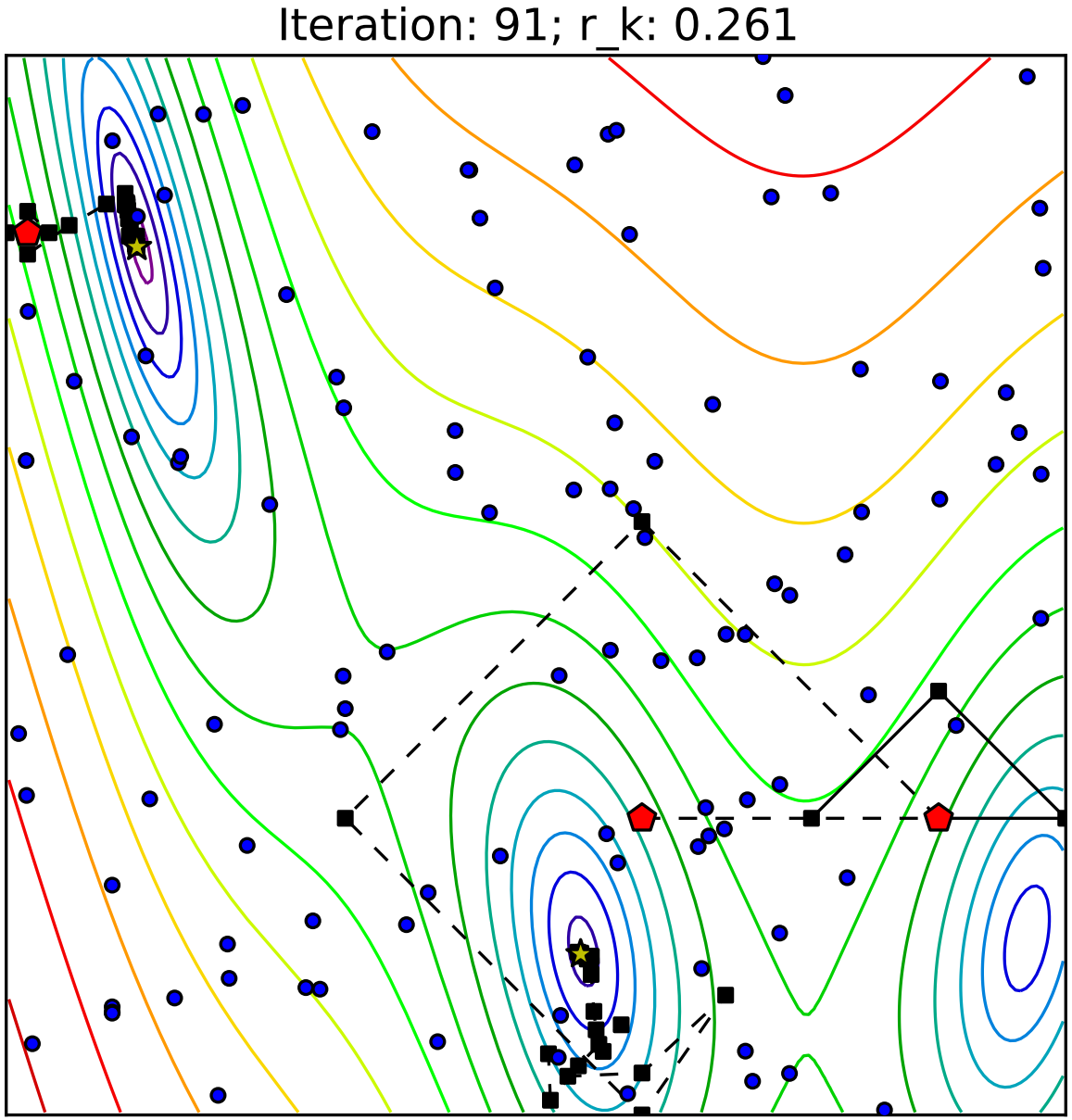
APOSMM



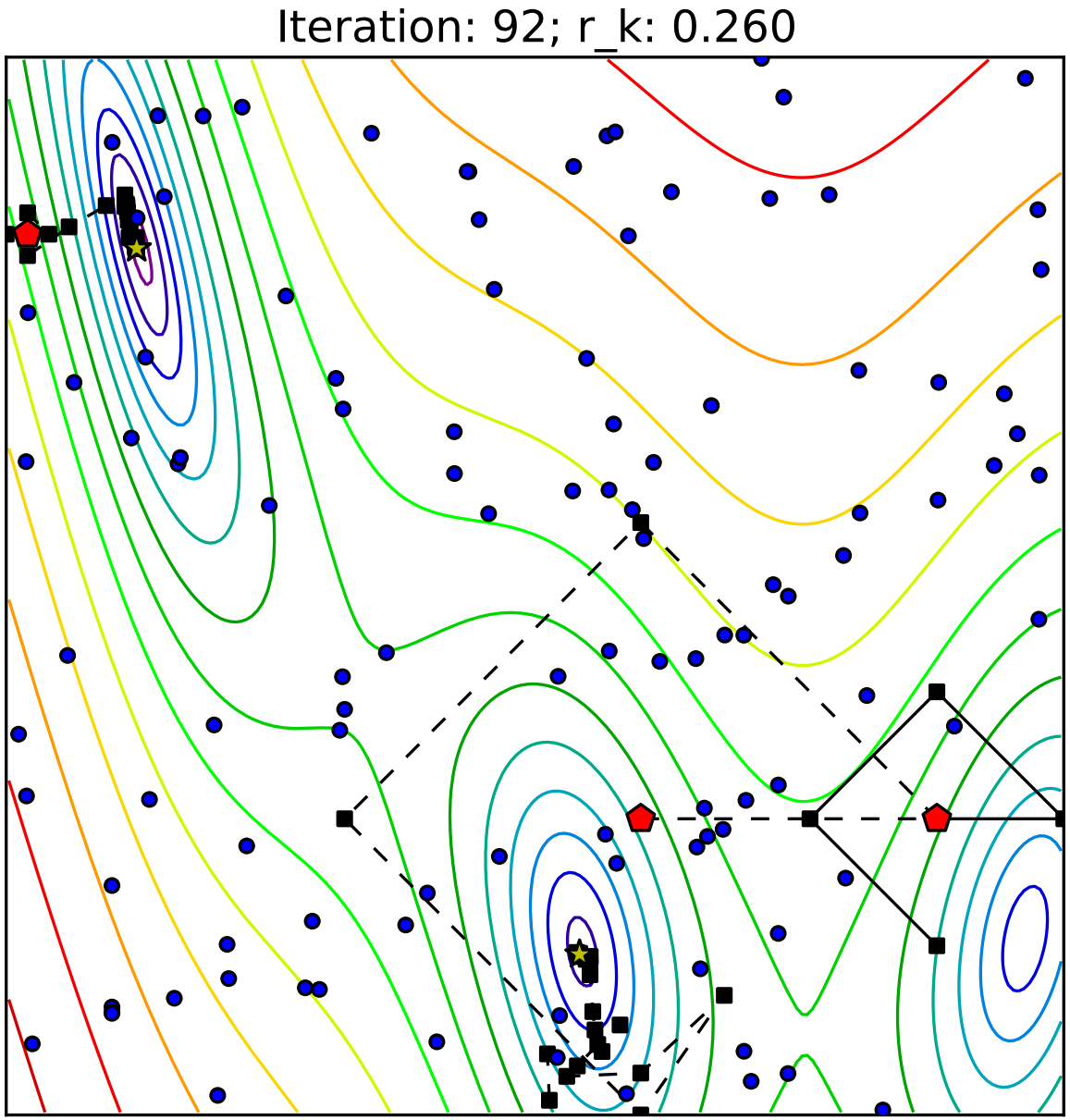
APOSMM



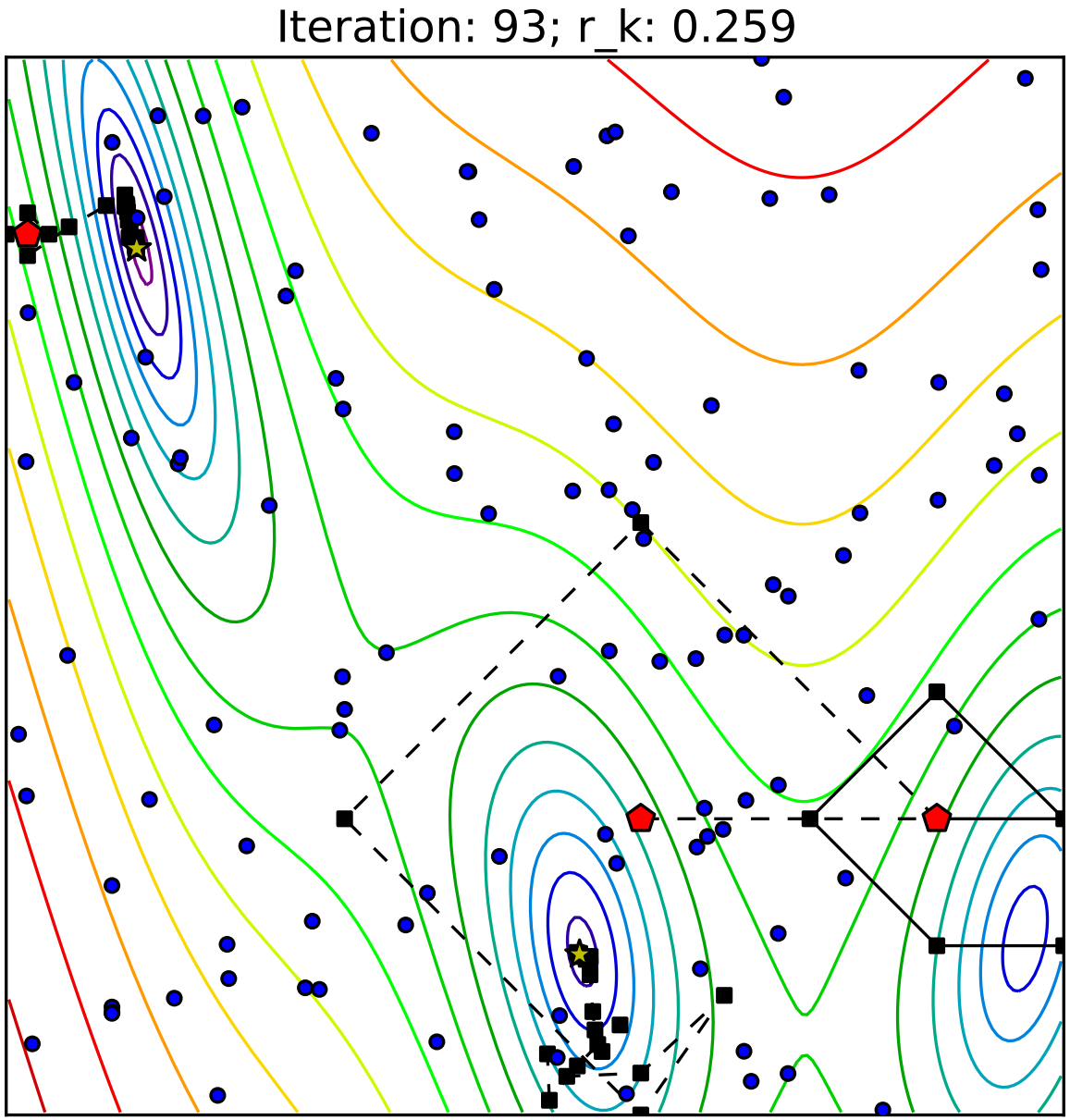
APOSMM



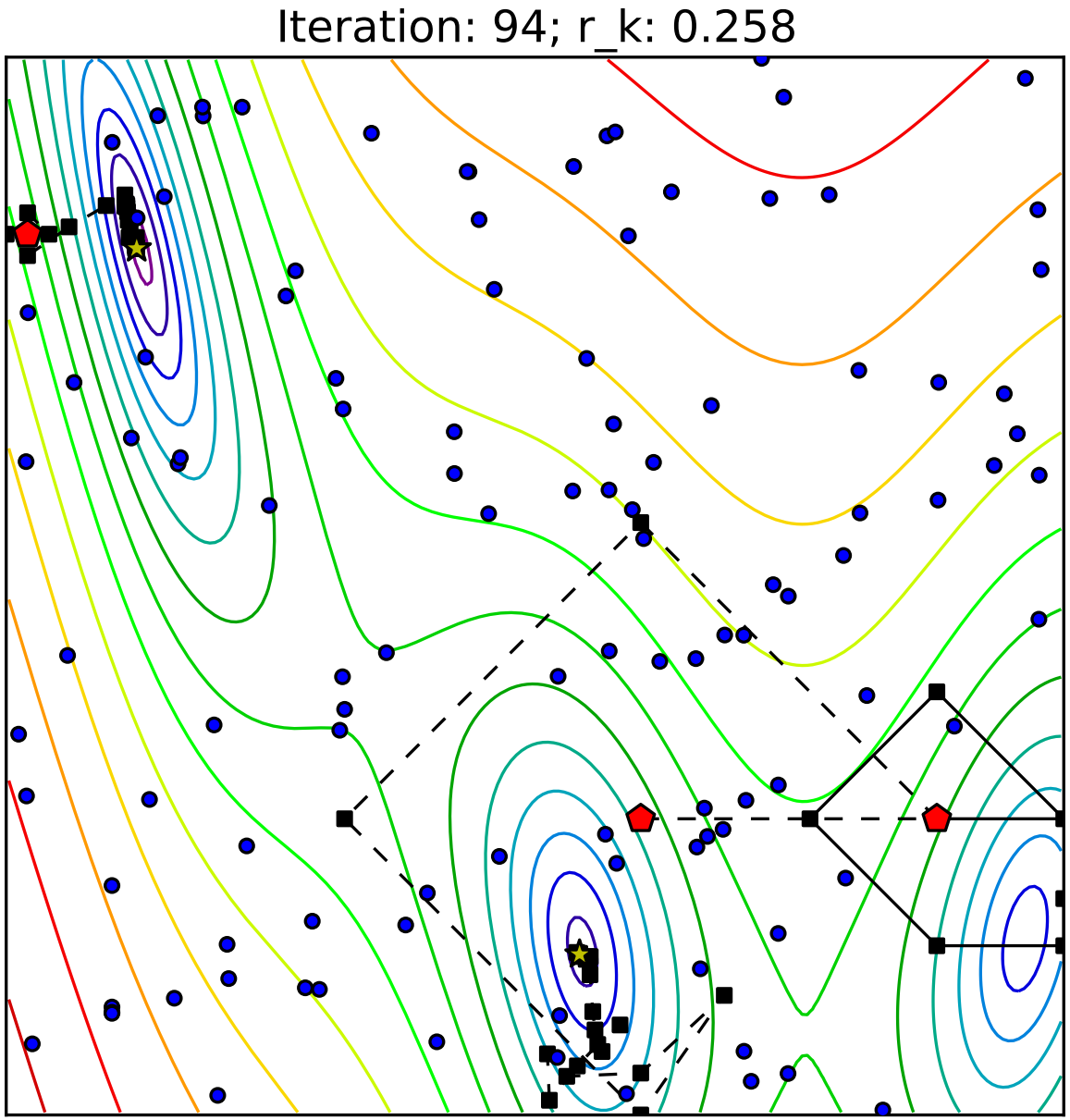
APOSMM



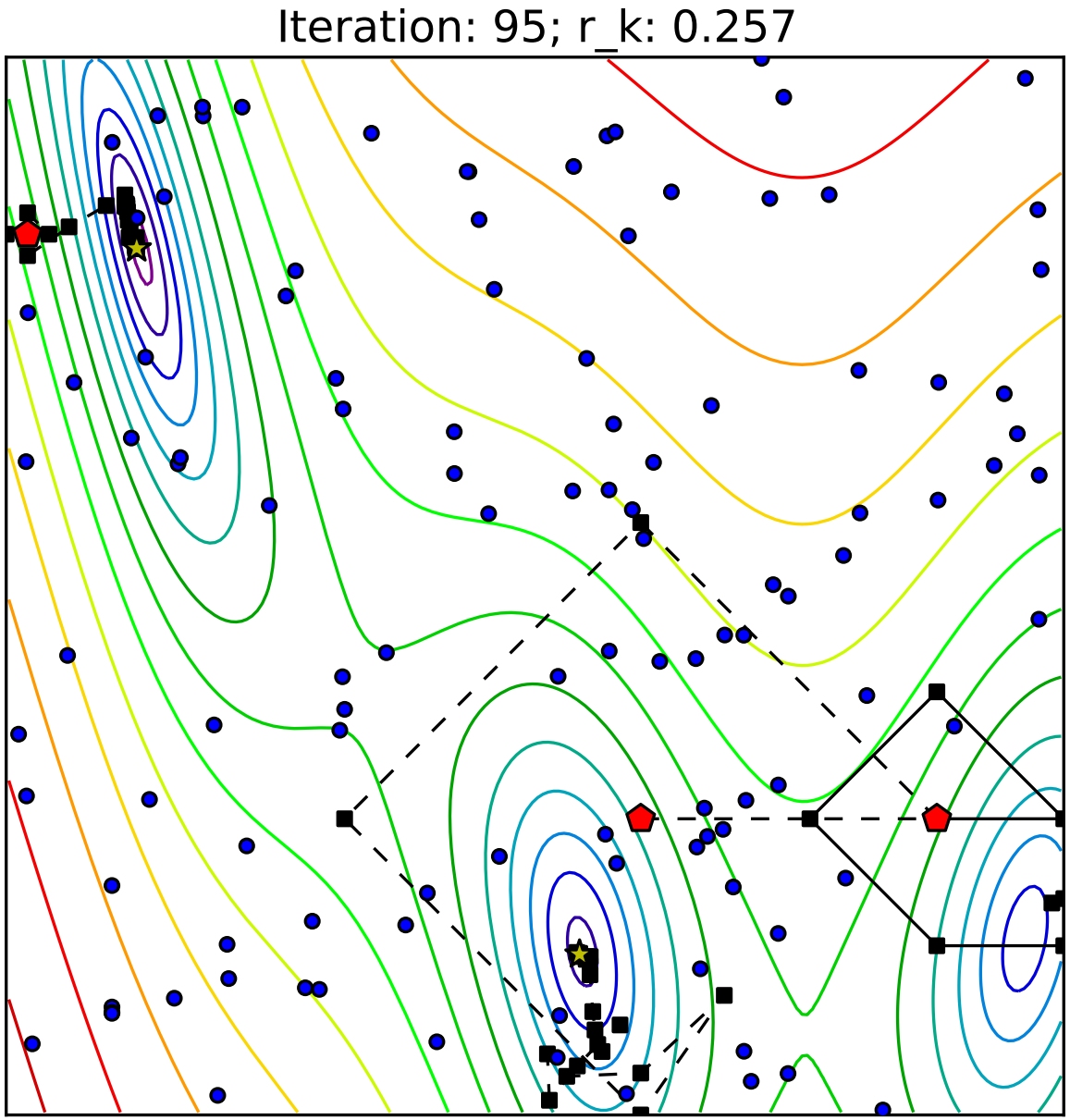
APOSMM



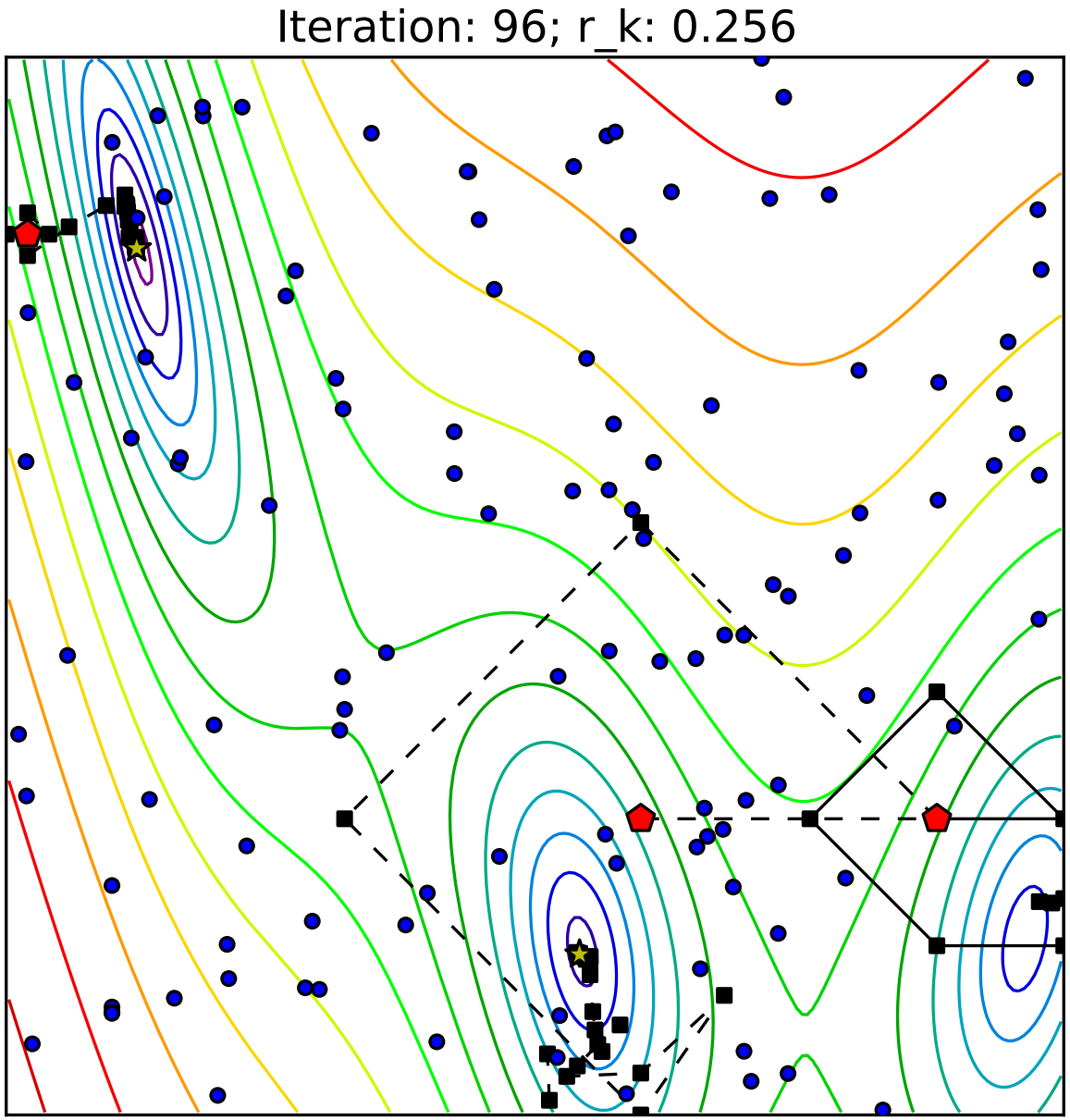
APOSMM



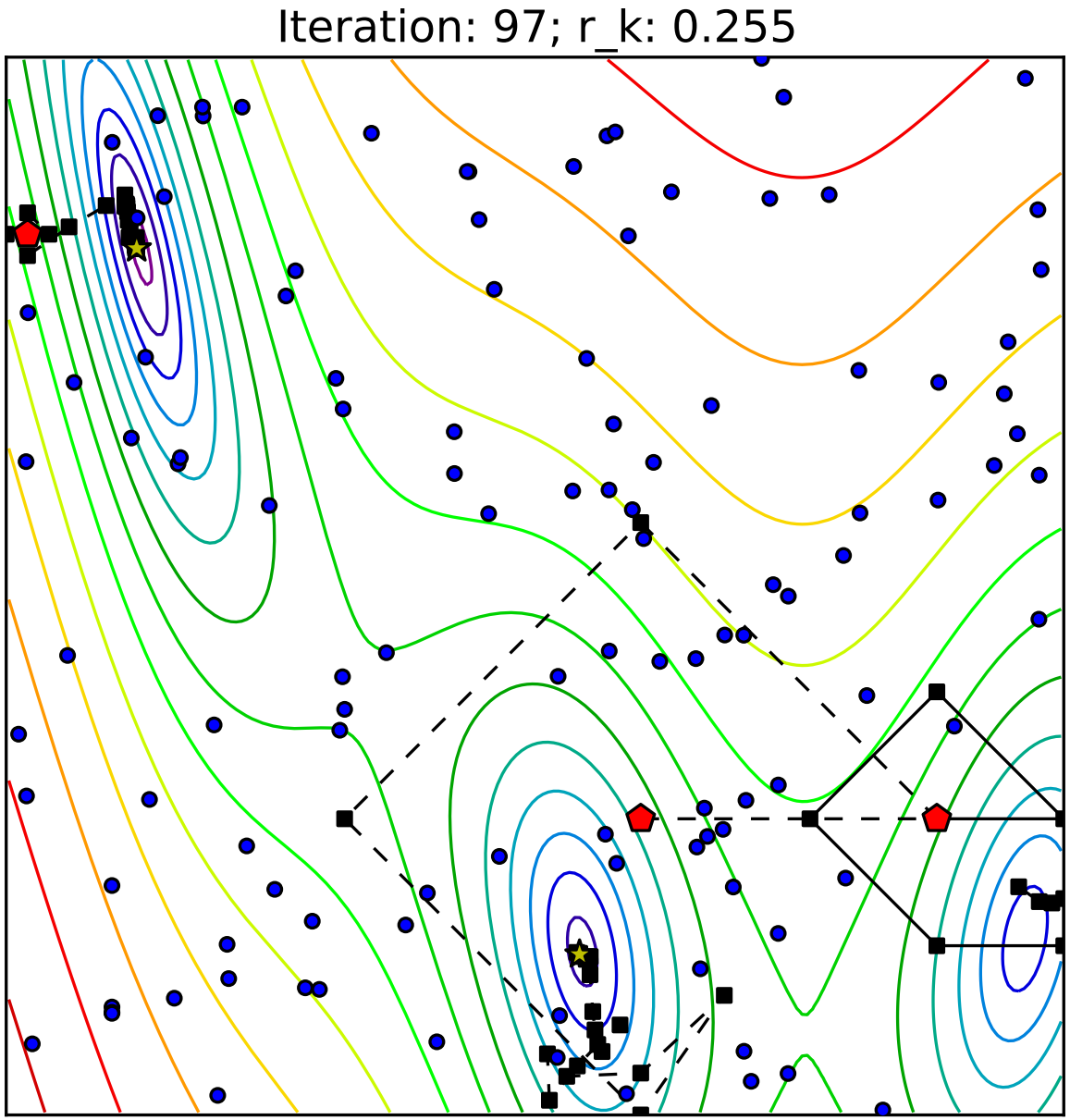
APOSMM



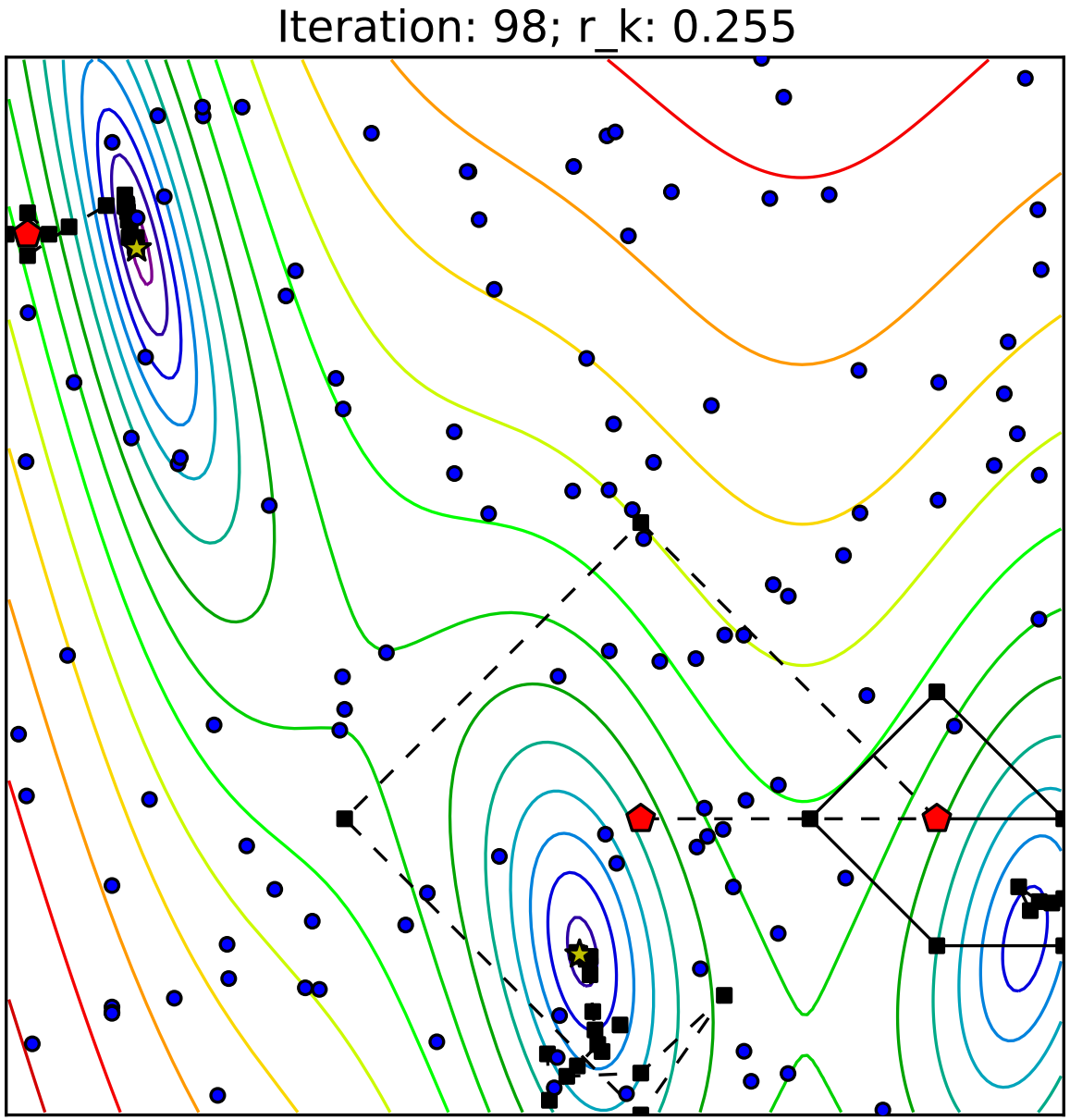
APOSMM



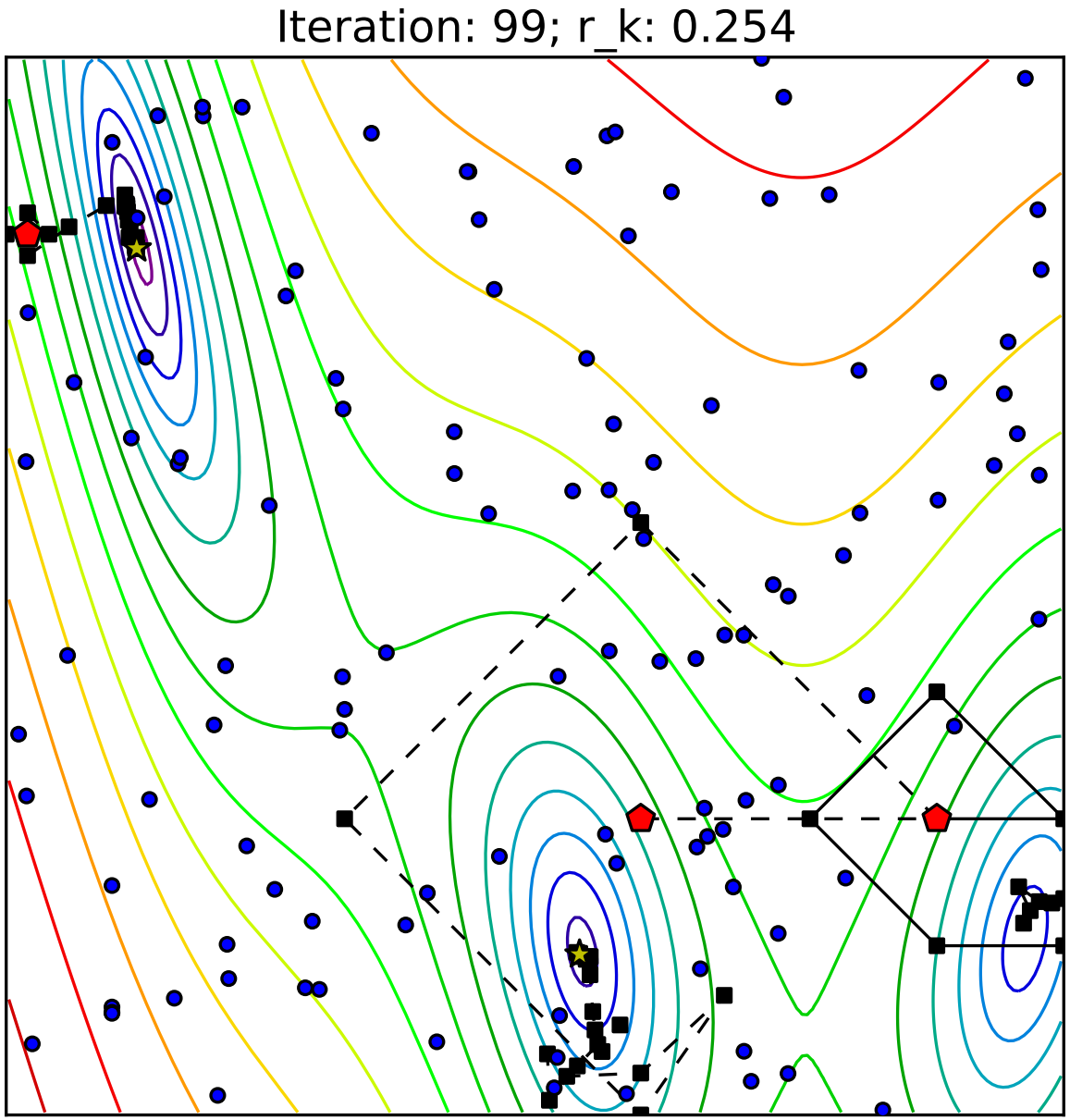
APOSMM



APOSMM



APOSMM



Properties of the local optimization method

Necessary:

- Honors a starting point
- Honors bound constraints

Properties of the local optimization method

Necessary:

- Honors a starting point
- Honors bound constraints

ORBIT satisfies these [Wild, Regis, Shoemaker, SIAM-JOSC, 2008]

BOBYQA satisfies these [Powell, 2009]

Properties of the local optimization method

Necessary:

- Honors a starting point
- Honors bound constraints

ORBIT satisfies these [Wild, Regis, Shoemaker, SIAM-JOSC, 2008]

BOBYQA satisfies these [Powell, 2009]

Possibly beneficial:

- Can return multiple points of interest
- Reports solution quality/confidence at every iteration
- Can avoid certain regions in the domain
- Uses a history of past evaluations of f
- Uses additional points mid-run

Measuring Performance

Notation:

Let X^* be the set of all local minima of f.
Let $f^*_{(i)}$ be the ith smallest value $\{f(x^*)|x^* \in X^*\}$.
Let $x^*_{(i)}$ be the element of X^* corresponding to the value $f^*_{(i)}$.

The global minimum has been found at a level $\tau > 0$ at batch k if an algorithm it has found a point $\hat{x}$ satisfying:

$$f(\hat{x}) - f^*_{(1)} \leq (1 - \tau)\left(f(x_0) - f^*_{(1)}\right),$$

where x_0 is the starting point for problem p.

Measuring Performance

Notation:

Let X^* be the set of all local minima of f.
Let $f^*_{(i)}$ be the ith smallest value $\{f(x^*)|x^* \in X^*\}$.
Let $x^*_{(i)}$ be the element of X^* corresponding to the value $f^*_{(i)}$.

The j best local minima have been found at a level $\tau > 0$ at batch k if:

$$\left|\left\{x^*_{(1)}, \ldots, x^*_{(\underline{j}-1)}\right\} \bigcap \left\{x^*_{(i)} : \exists x \in \mathcal{H}_k \text{ with } \left\|x - x^*_{(i)}\right\| \leq r_n(\tau)\right\}\right| = \underline{j} - 1$$

&

$$\left|\left\{x^*_{(\underline{j})}, \ldots, x^*_{(\bar{j})}\right\} \bigcap \left\{x^*_{(i)} : \exists x \in \mathcal{H}_k \text{ with } \left\|x - x^*_{(i)}\right\| \leq r_n(\tau)\right\}\right| \geq j - \underline{j} + 1,$$

where $\underline{j}$ and $\bar{j}$ are the smallest and largest integers such that $f^*_{(\bar{j})} = f^*_{(j)} = f^*_{(\underline{j})}$ and where $r_n(\tau) = \sqrt[n]{\frac{\tau \, \text{vol}(\mathcal{D})\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$.

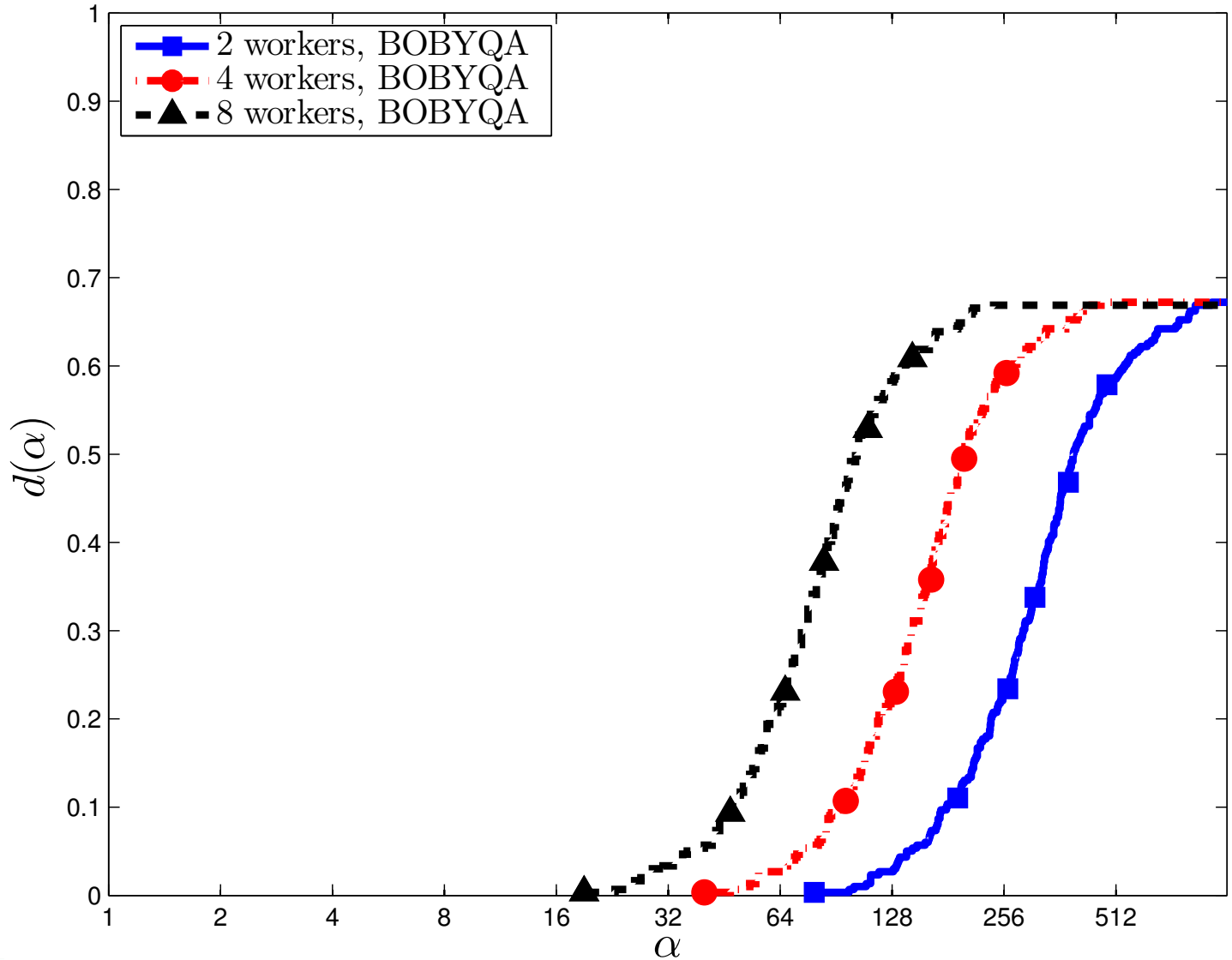
Problems considered

GKLS problem generator [Gaviano et al., "Algorithm 829" (TOMS, 2003)]

- 600 synthetic problems with known local minima
- $n = 2, \ldots, 7$
- 10 local minima in the unit cube with a unique global minimum
- 100 problems for each dimension
- 5 replications (different seeds) for each problem
- 5000 evaluations

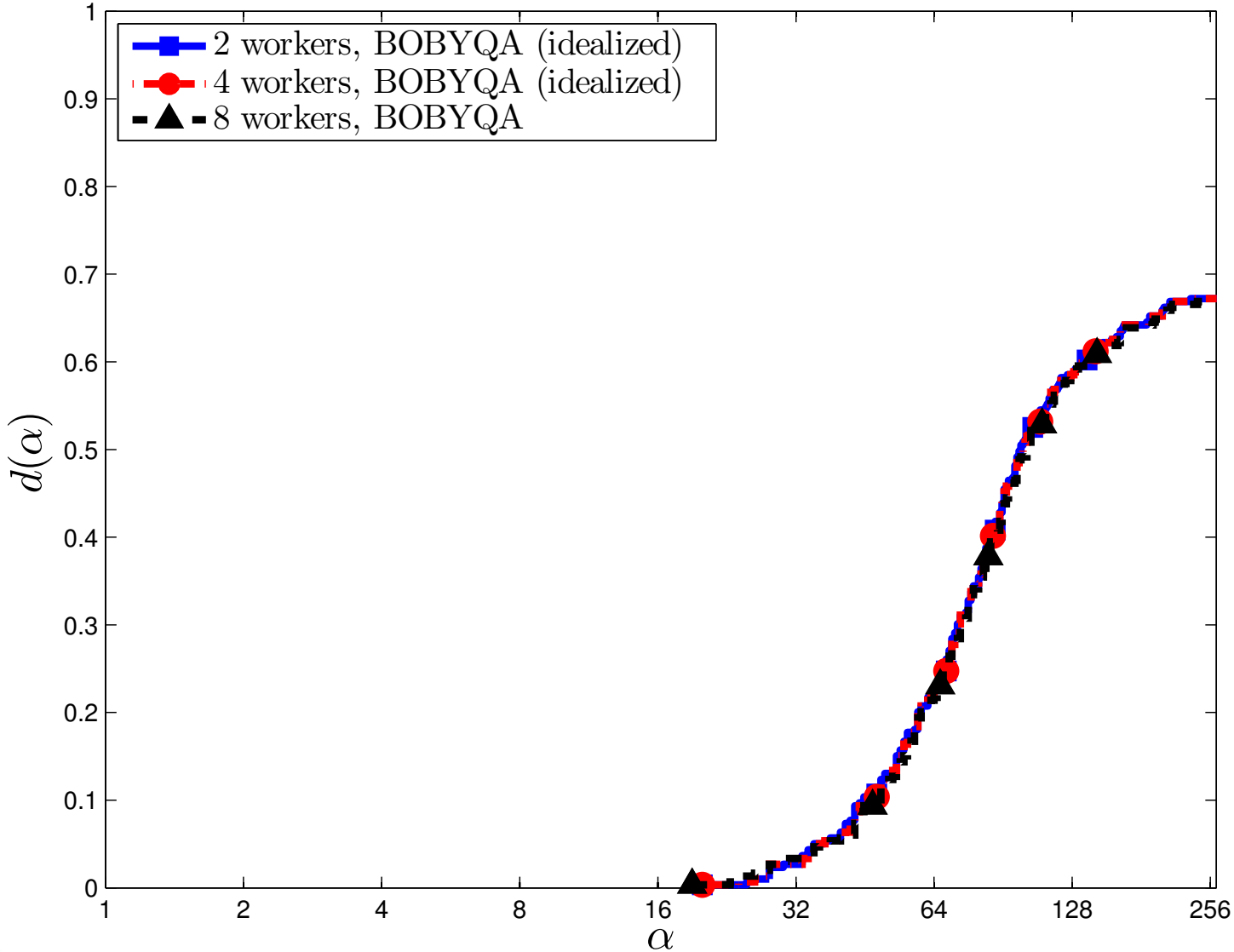
Data Profiles

Within $\sqrt[n]{\frac{10^{-3}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 7 best minima



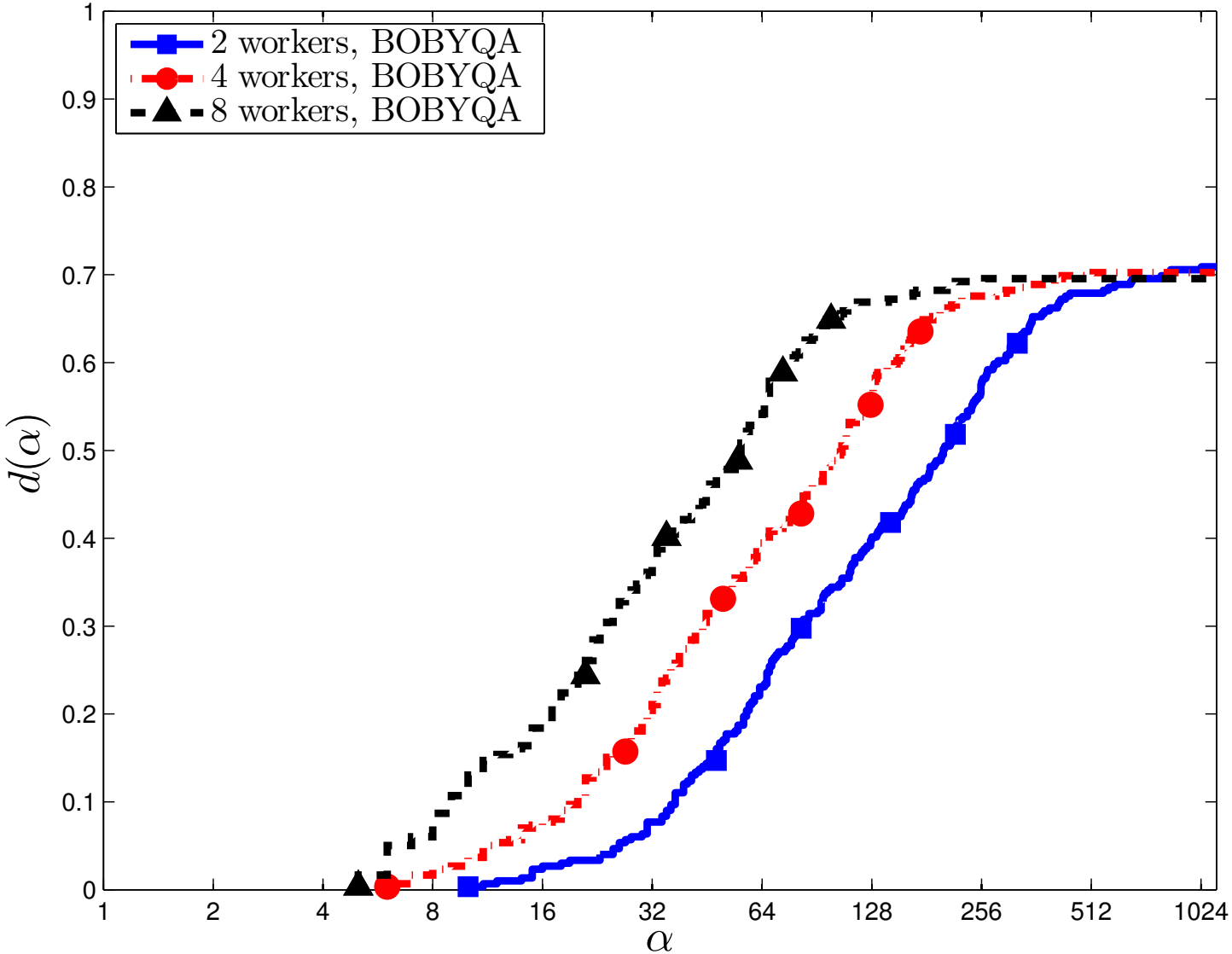
Data Profiles

Within $\sqrt[n]{\frac{10^{-3}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 7 best minima



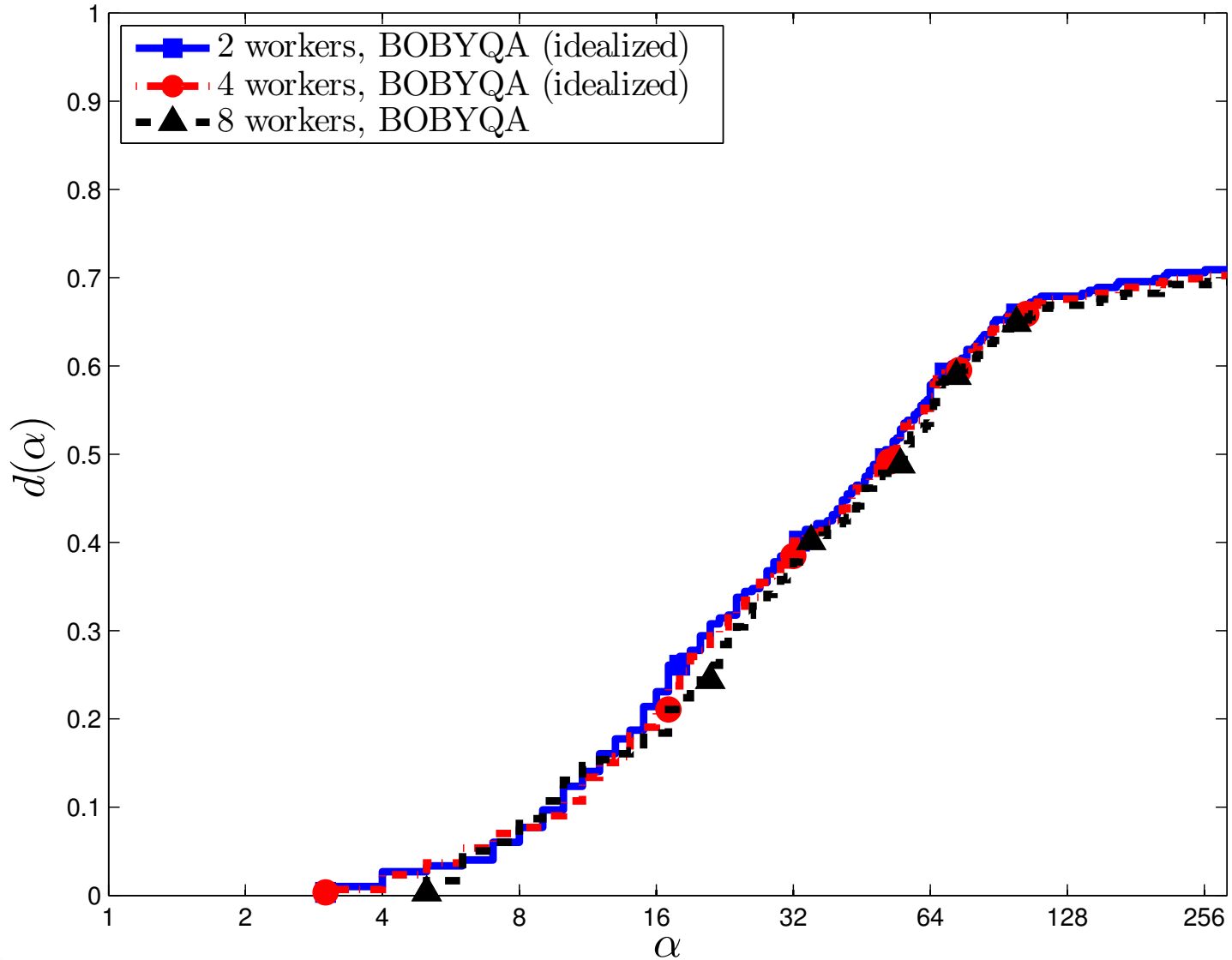
Data Profiles

Within $\sqrt[n]{\frac{10^{-4}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 3 best minima



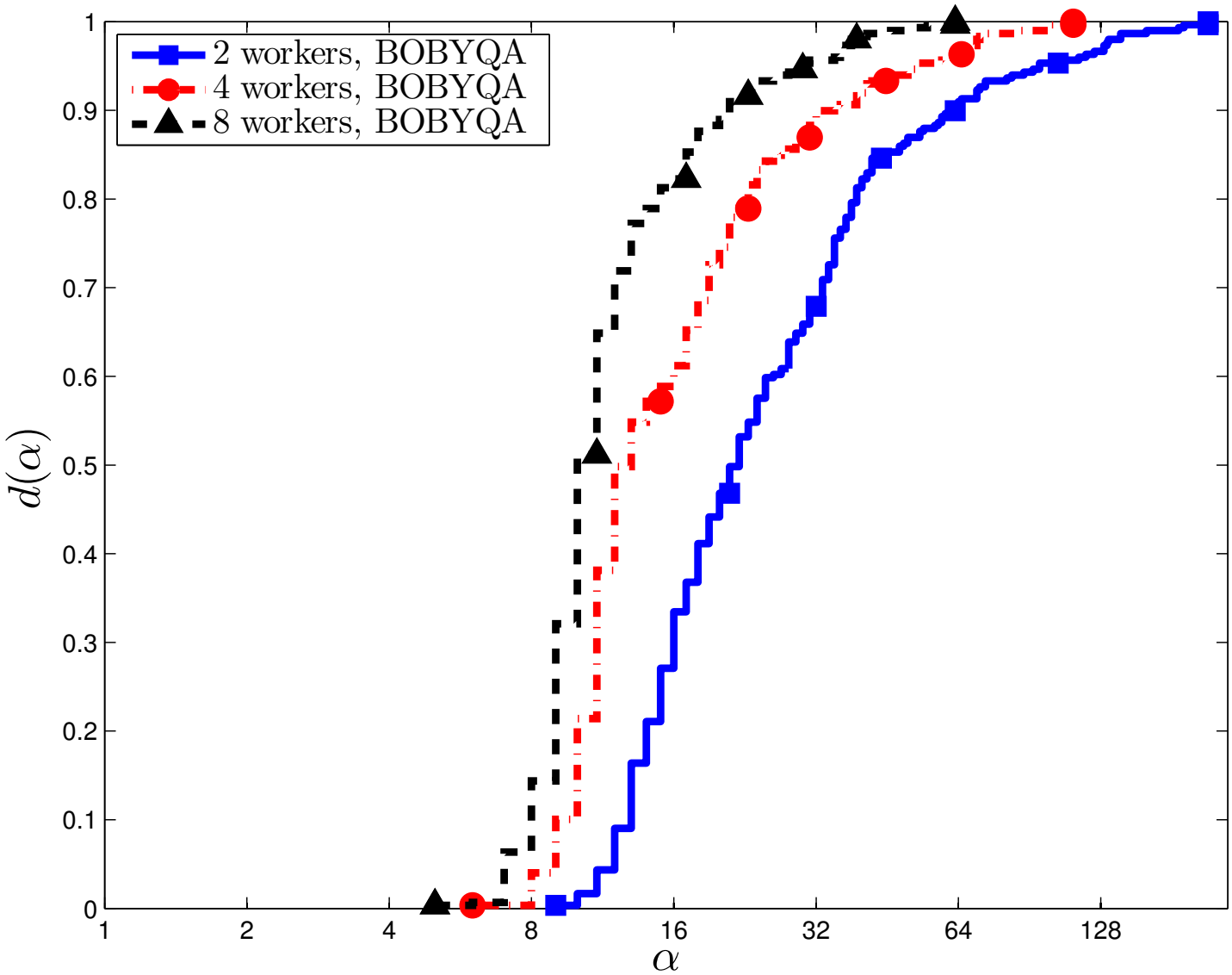
Data Profiles

Within $\sqrt[n]{\frac{10^{-4}\Gamma(\frac{n}{2}+1)}{\pi^{n/2}}}$ of 3 best minima



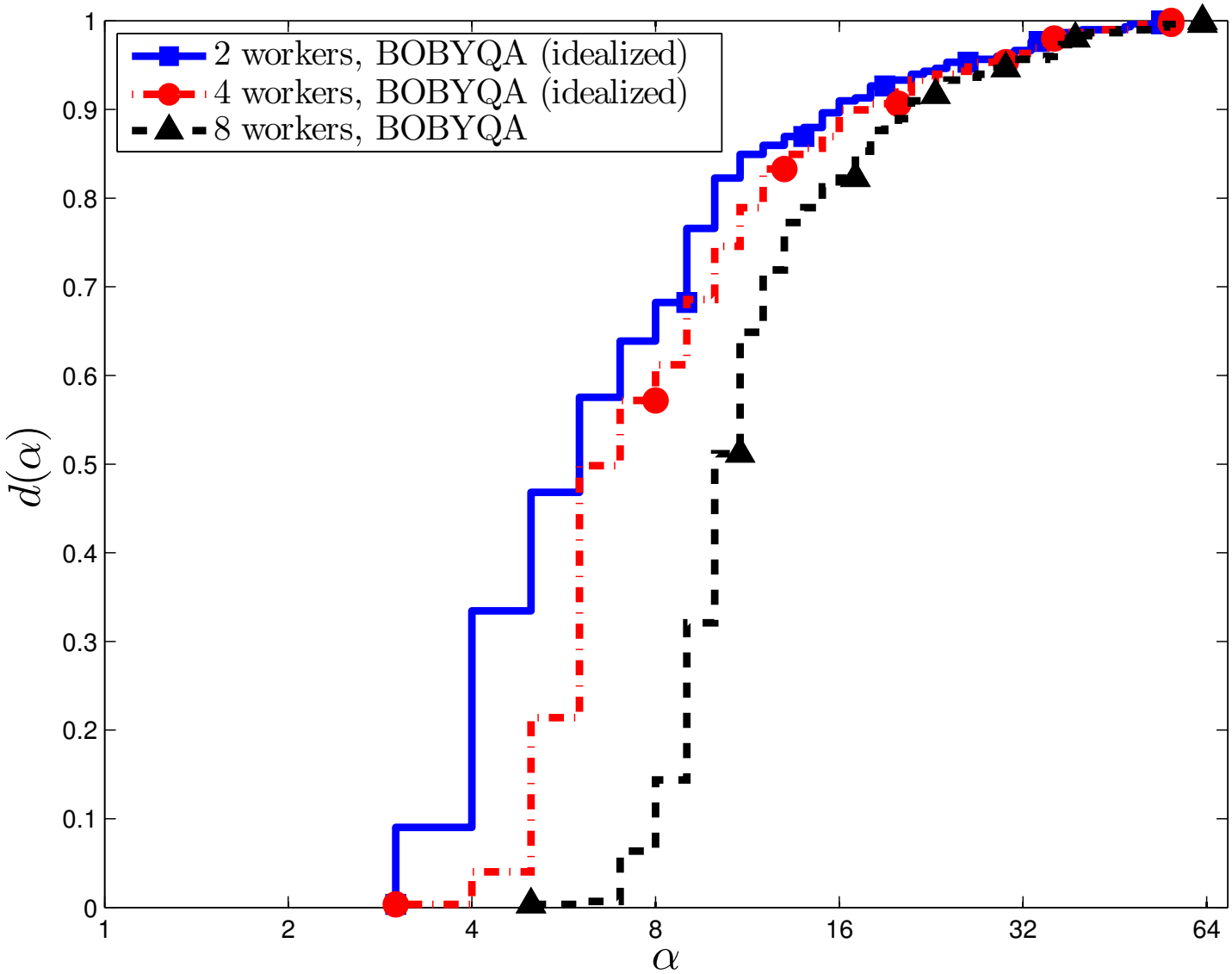
Data Profiles

$$f(x) - f^*_{(1)} \leq (1 - 10^{-5}) \left(f(x_0) - f^*_{(1)} \right)$$



Data Profiles

$$f(x) - f^*_{(1)} \leq (1 - 10^{-5}) \left(f(x_0) - f^*_{(1)} \right)$$



Closing Remarks

- `libEnsemble` is a software library to coordinate the concurrent evaluation of calculations.
- We have a growing set of use cases and examples.
- Let us know if you have examples you'd like to see.